

The recursion hierarchy for PCF is strict

John Longley

July 18, 2016

Abstract

Let PCF_k denote the sublanguage of Plotkin's PCF in which fixed point operators Y_σ are admitted only for types σ of level $\leq k$. We show that the languages PCF_k form a strict hierarchy, in the sense that none of the Y_σ for σ of level $k+1$ are definable in PCF_k up to observational equivalence. This answers a question posed by Berger in 1999. Our proof makes substantial use of the theory of *nested sequential procedures* (also called PCF *Böhm trees*) as expounded in the recent book of Longley and Normann.

1 Introduction

In this paper we study sublanguages of Plotkin's functional programming language PCF, which we here take to be the simply typed λ -calculus over a single base type \mathbb{N} , with constants

$$\begin{array}{ll} \hat{n} & : \mathbb{N} \text{ for each } n \in \mathbb{N}, \quad \text{succ, pre} & : \mathbb{N} \rightarrow \mathbb{N}, \\ \text{ifzero} & : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}, \quad Y_\sigma & : (\sigma \rightarrow \sigma) \rightarrow \sigma \text{ for each type } \sigma. \end{array}$$

As usual, we will consider this language to be endowed with a certain (call-by-name) operational semantics, which in turn gives rise to a notion of *observational equivalence* for PCF programs.

We define the *level* $\text{lv}(\sigma)$ of a type σ inductively by

$$\text{lv}(\mathbb{N}) = 0, \quad \text{lv}(\sigma \rightarrow \tau) = \max(\text{lv}(\sigma) + 1, \text{lv}(\tau)),$$

and define the *pure type* \bar{k} of level $k \in \mathbb{N}$ by

$$\bar{0} = \mathbb{N}, \quad \overline{k+1} = \bar{k} \rightarrow \mathbb{N}.$$

Modifying the definition of PCF so that the constants Y_σ are admitted only for types σ of level $\leq k$, we obtain a sublanguage PCF_k for any $k \in \mathbb{N}$. Our main result will be that for each k , the expressive power of PCF_{k+1} strictly exceeds that of PCF_k : in particular, there is no closed term of PCF_k that is observationally equivalent to $Y_{\overline{k+1}}$. (Note that ‘observational equivalence’ has the same meaning for all the languages of interest here, as will be explained

in Section 2.) This answers a question posed explicitly by Berger in [4], but present in the folklore at least since the early 1990s. It is worth noting that the situation is quite different in several extensions of PCF considered in the literature, in which one can restrict to recursions at level 1 types without loss of expressivity (see the end of Subsection 2.4).

Our discussion will focus on two models of PCF, both studied in detail in the recent book of Longley and Normann [14]: the *nested sequential procedure* model \mathbf{SP}^0 (also known in the literature as the PCF *Böhm tree* model and by various other names), and its extensional quotient \mathbf{SF} consisting of *sequential functionals*. These have effective submodels $\mathbf{SP}^{0,\text{eff}}$ and \mathbf{SF}^{eff} respectively. It is well-known that \mathbf{SF} and \mathbf{SF}^{eff} are fully abstract for PCF, and indeed that \mathbf{SF}^{eff} is isomorphic to the closed term model of PCF modulo observational equivalence. Our result can therefore be understood more denotationally as saying that more elements of \mathbf{SF}^{eff} are denotable in PCF_{k+1} than in PCF_k . (It follows easily that the same holds with \mathbf{SF}^{eff} replaced by any other adequate, compositional model of PCF, such as the Scott model.) From this we may also infer that there is no finite ‘basis’ $B \subseteq \mathbf{SF}^{\text{eff}}$ relative to which all elements of \mathbf{SF}^{eff} are λ -definable (see Corollary 3 below).

In Section 2 we recall the necessary technical background on PCF and on the models \mathbf{SP}^0 and \mathbf{SF} , fleshing out the ideas outlined informally above. In Section 3 we obtain a convenient inductive characterization of the class of procedures definable in (the ‘oracle’ version of) PCF_k , framed in terms of constructions on the procedures themselves. In Section 4 we introduce the concept of a *spinal* procedure term, and show using our inductive characterization that no PCF_k -denotable procedure can be spinal (this is the most demanding part of the proof). Since the standard interpretation of $Y_{\overline{k+1}}$ in \mathbf{SP}^0 is spinal, this already shows that the element $Y_{\overline{k+1}} \in \mathbf{SP}^0$ is not definable in PCF_k . However, this still leaves open the possibility that there might be other NSPs, distinct from Y but extensionally equivalent to it, that are denotable in PCF_k . We will show in Section 5 that this is not the case, at least if we consider $Y_{\mathbb{N} \rightarrow \overline{k+1}}$ in place of $Y_{\overline{k+1}}$; we therefore have an element of \mathbf{SF} -denotable in PCF_{k+1} but not in PCF_k , establishing Berger’s conjecture. In Section 6 we refine our methods slightly to show that even $Y_{\overline{k+1}} \in \mathbf{SF}$ is not PCF_k -denotable, so that in fact *none* of the operators Y_σ with $\text{lv}(\sigma) = k+1$ are definable in PCF_k . We conclude in Section 7 with a discussion of related and future work.

I am grateful to Ulrich Berger, Martín Escardó, Dag Normann and Alex Simpson for valuable discussions and correspondence, and to Colin Stirling for drawing my attention to the related work of Damm and Statman (as discussed in Section 7). Many of the participants in the Domains XII workshop in Cork and the Galop XI workshop in Eindhoven also offered valuable comments.

The present paper is a revised, corrected and expanded of a University of Edinburgh technical report from July 2015, the most significant changes being: the addition of the material on the pure type $\overline{k+1}$ in Section 6; a simplified approach to characterizing a suitable substructure of \mathbf{SP}^0 in Section 3; and some formal tightening of the material in Section 2.2 and the proof of Lemma 20.

2 Background

We here summarize the necessary definitions and technical background from [14], especially from Chapters 6 and 7.

2.1 The language PCF

In [21], Scott introduced the language LCF for computable functionals of simple type. This language is traditionally called PCF when equipped with a standalone operational semantics as in Plotkin [18]. We will work here with the same version of PCF as in [14], with the natural numbers as the only base type. Our types σ are thus generated by

$$\sigma ::= \mathbf{N} \mid \sigma \rightarrow \sigma ,$$

and our terms will be those of the simply typed λ -calculus constructed from the constants

$$\begin{aligned} \widehat{n} &: \mathbf{N} && \text{for each } n \in \mathbb{N} , \\ \text{succ}, \text{pre} &: \mathbf{N} \rightarrow \mathbf{N} , \\ \text{ifzero} &: \mathbf{N} \rightarrow \mathbf{N} \rightarrow \mathbf{N} \rightarrow \mathbf{N} , \\ Y_\sigma &: (\sigma \rightarrow \sigma) \rightarrow \sigma && \text{for each type } \sigma . \end{aligned}$$

We often abbreviate the type $\sigma_0 \rightarrow \dots \rightarrow \sigma_{r-1} \rightarrow \mathbf{N}$ to $\sigma_0, \dots, \sigma_{r-1} \rightarrow \mathbf{N}$ or just $\vec{\sigma} \rightarrow \mathbf{N}$. As usual, we write $\Gamma \vdash M : \sigma$ to mean that M is a well-typed term in the environment Γ (where Γ is a finite list of typed variables). Throughout the paper, we shall regard the type of a variable x as intrinsic to x , and will often write x^σ to indicate that x carries the type σ . For each $k \in \mathbb{N}$, the sublanguage PCF_k is obtained by admitting the constants Y_σ only for types σ of level $\leq k$.

We endow the class of closed PCF terms with the following small-step reduction rules:

$$\begin{aligned} (\lambda x.M)N &\rightsquigarrow M[x \mapsto N] , & \text{ifzero } \widehat{0} &\rightsquigarrow \lambda xy.x , \\ \text{succ } \widehat{n} &\rightsquigarrow \widehat{n+1} , & \text{ifzero } \widehat{n+1} &\rightsquigarrow \lambda xy.y , \\ \text{pre } \widehat{n+1} &\rightsquigarrow \widehat{n} , & Y_\sigma M &\rightsquigarrow M(Y_\sigma M) . \\ \text{pre } \widehat{0} &\rightsquigarrow \widehat{0} , \end{aligned}$$

These reductions may be applied in any *evaluation context*—that is, a context generated by composition from the basic contexts

$$[-]N , \quad \text{succ } [-] , \quad \text{pre } [-] , \quad \text{ifzero } [-] .$$

Thus, the relation \rightsquigarrow is generated by the rules above along with the clause: if $M \rightsquigarrow M'$ and $E[-]$ is an evaluation context, then $E[M] \rightsquigarrow E[M']$. We write \rightsquigarrow^* for the reflexive-transitive closure of \rightsquigarrow . If Q is any closed PCF term of type \mathbf{N} , it is easy to check that either $Q \rightsquigarrow^* \widehat{n}$ for some $n \in \mathbb{N}$ or the reduction sequence starting from Q is infinite.

This completes the definition of the languages PCF and PCF_k . Whilst the language PCF_0 is too weak for programming purposes (it cannot even define addition), it is not hard to show that even PCF_1 is Turing-complete: that is, any partial computable function $\mathbb{N} \rightarrow \mathbb{N}$ is representable by a closed PCF_1 term of type $\mathbb{N} \rightarrow \mathbb{N}$.

We will also refer to the non-effective language PCF^Ω (or *oracle* PCF) obtained by extending the definition of PCF with a constant $C_f : \mathbb{N} \rightarrow \mathbb{N}$ for every set-theoretic partial function $f : \mathbb{N} \rightarrow \mathbb{N}$, along with a reduction rule $C_f \hat{n} \rightsquigarrow \hat{m}$ for every n, m such that $f(n) = m$. (In PCF^Ω , the evaluation of a closed term $Q : \mathbb{N}$ may fail to reach a value \hat{n} either because it generates an infinite computation, or because it encounters a subterm $C_f(n)$ where $f(n)$ is undefined.) The languages PCF_k^Ω are defined analogously.

If M, M' are closed PCF^Ω terms of the same type σ , and \mathcal{L} is one of our languages $\text{PCF}_k, \text{PCF}_k^\Omega, \text{PCF}, \text{PCF}^\Omega$, we say that M, M' are *observationally equivalent in \mathcal{L}* , and write $M \simeq_{\mathcal{L}} M'$, if for all closed program contexts $C[-^\sigma] : \mathbb{N}$ of \mathcal{L} and all $n \in \mathbb{N}$, we have

$$C[M] \rightsquigarrow^* n \text{ iff } C[M'] \rightsquigarrow^* n.$$

Fortunately, it is easy to see that all of the above languages give rise to exactly the same relation $\simeq_{\mathcal{L}}$. First, it is immediate from the definition that if $\mathcal{L}, \mathcal{L}'$ are two of our languages and $\mathcal{L} \supseteq \mathcal{L}'$, then $\simeq_{\mathcal{L}} \subseteq \simeq_{\mathcal{L}'}$. It therefore only remains to show that $M \simeq_{\text{PCF}_0} M'$ implies $M \simeq_{\text{PCF}^\Omega} M'$. For this, we use the fact that any of the constants Y_σ or C_f in PCF^Ω can be ‘approximated’ to any desired accuracy by terms of PCF_0 . Indeed, for any $j \in \mathbb{N}$, we may define PCF_0 terms

$$\begin{aligned} Y_\sigma^{(j)} &= \lambda f^{\sigma \rightarrow \sigma}. f^j(\lambda \vec{x}. \perp), \\ C_f^{(j)} &= \lambda n. \text{case } n \text{ of } (0 \Rightarrow \widehat{f(0)} \mid \cdots \mid j-1 \Rightarrow \widehat{f(j-1)}), \end{aligned}$$

writing \perp for $Y_0(\lambda x^0. x)$, and using some evident syntactic sugar in the definition of $C_f^{(j)}$. For any PCF^Ω term M , let $M^{(j)}$ denote the ‘approximation’ obtained from M by replacing all occurrences of constants Y_σ, C_f by $Y_\sigma^{(j)}, C_f^{(j)}$ respectively. It is then not hard to show that for closed $Q : \mathbb{N}$, we have

$$Q \rightsquigarrow^* \hat{n} \text{ iff } \exists j. Q^{(j)} \rightsquigarrow^* \hat{n}.$$

From this it follows easily that if $C[-]$ is an observing context of PCF^Ω that distinguishes M, M' , then some approximation $C^{(j)}[-]$ (a context of PCF_0) also suffices to distinguish them. This establishes that $\simeq_{\text{PCF}_0} \subseteq \simeq_{\text{PCF}^\Omega}$. We may therefore write \simeq for observational equivalence without ambiguity.

In fact, an even more restricted class of observing contexts suffices for ascertaining observational equivalence of PCF^Ω terms. The well-known *context lemma*, due to Milner [15], states that $M \simeq M' : \sigma_0, \dots, \sigma_{r-1} \rightarrow \mathbb{N}$ iff M, M' have the same behaviour in all *applicative contexts* of PCF—that is, if for all closed PCF terms $N_0 : \sigma_0, \dots, N_{r-1} : \sigma_{r-1}$, we have

$$MN_0 \dots N_{r-1} \rightsquigarrow^* n \text{ iff } M'N_0 \dots N_{r-1} \rightsquigarrow^* n.$$

Furthermore, using the above idea of approximation, it is easy to see that we obtain exactly the same equivalence relation if we allow the N_i here to range only over closed PCF_0 terms—this gives us the notion of PCF_0 *applicative equivalence*, which we shall denote by \sim_0 .

We have concentrated so far on giving a purely operational description of PCF. We are now able to express the operational content of our main theorem as follows. As in Section 1, we define the type k by $\bar{0} = \mathbb{N}$, $k + 1 = \bar{k} \rightarrow \mathbb{N}$; we shall write \bar{k} simply as k where there is no risk of confusion.

Theorem 1 *For any $k \geq 1$, there are functionals definable in PCF_{k+1} but not in PCF_k^Ω . More precisely, there is no closed term M of PCF_k^Ω such that $M \simeq Y_{k+1}$ (or equivalently $M \sim_0 Y_{k+1}$).*

In fact, we shall first obtain a version of Theorem 1 with $Y_{\mathbb{N} \rightarrow (k+1)}$ in place of Y_{k+1} , then resort to a more indirect method obtain the result for Y_{k+1} itself in Section 6.

Theorem 1 can be construed as saying that in a suitably pure fragment of a functional language such as Haskell, the computational strength of recursive function definitions increases strictly as the admissible type level for such recursions is increased. The point of the formulation in terms of \sim_0 is to present our result in a manifestly strong form: there is no $M \in \text{PCF}_k$ that induces the same partial function as $Y_{\mathbb{N} \rightarrow (k+1)}$ even on closed PCF_0 terms.

A slightly more denotational formulation of our theorem can be given in terms of the model **SF** of *sequential functionals*, which we may here define as the type structure of closed PCF^Ω terms modulo observational equivalence. Specifically, for each type σ , let $\text{SF}(\sigma)$ denote the set of closed PCF^Ω terms $M : \sigma$ modulo \simeq . Clearly, application of PCF^Ω terms induces a well-defined function $\cdot : \text{SF}(\sigma \rightarrow \tau) \times \text{SF}(\sigma) \rightarrow \text{SF}(\tau)$ for any σ, τ ; the structure **SF** then consists of the sets $\text{SF}(\sigma)$ along with these application operations. Using the context lemma, it is easy to see that $\text{SF}(\mathbb{N}) \cong \mathbb{N}_\perp (= \mathbb{N} \sqcup \{\perp\})$ and that $\text{SF}(\sigma \rightarrow \tau)$ is isomorphic to a set of *functions* $\text{SF}(\sigma) \rightarrow \text{SF}(\tau)$: that is, if $f, f' \in \text{SF}(\sigma \rightarrow \tau)$ satisfy $f \cdot x = f' \cdot x$ for all $x \in \text{SF}(\sigma)$, then $f = f'$.

Any closed PCF^Ω term $M : \sigma$ naturally has a denotation $\llbracket M \rrbracket$ in $\text{SF}(\sigma)$, namely its own equivalence class. We may therefore restate Theorem 1 as:

Theorem 2 *For any $k \geq 1$, the element $\llbracket Y_{k+1} \rrbracket \in \text{SF}$ is not denotable in PCF_k^Ω .*

It follows immediately that in any other adequate, compositional model of PCF^Ω (such as Scott's continuous model or Berry's stable model), the element $\llbracket Y_{\mathbb{N} \rightarrow (k+1)} \rrbracket$ is not PCF_k^Ω -definable, since the equivalence relation on PCF^Ω terms induced by such a model must be contained within \simeq .

By taking closed terms of PCF rather than PCF^Ω modulo observational equivalence, we obtain the type structure **SF**^{eff} of *effective sequential functionals*, which can clearly be seen as a substructure of **SF**. Although the above constructions of **SF** and **SF**^{eff} are syntactic, there are other more mathematical constructions (for instance, involving game models [1, 7]) that also give rise to these structures, and experience suggests that they are mathematically natural

classes of higher-order functionals. We now see that Theorem 2 implies an interesting absolute property of \mathbf{SF}^{eff} , not dependent on any choice of presentation for this structure or any selection of language primitives:

Corollary 3 (No finite basis) *There is no finite set B of elements of \mathbf{SF}^{eff} such that all elements of \mathbf{SF}^{eff} are λ -definable relative to B . In other words, the cartesian category of PCF-computable functionals is not finitely generated.*

PROOF Suppose $B = \{b_0, \dots, b_{n-1}\}$ were such a set. For each i , take a closed PCF term M_i denoting b_i . Then the terms M_0, \dots, M_{n-1} between them contain only finitely many occurrences of constants Y_σ , so these constants are all present in PCF_k for large enough k . But this means that b_0, \dots, b_{n-1} , and hence all elements of \mathbf{SF}^{eff} , are PCF_k -definable, contradicting Theorem 2. \square

2.2 The model SP^0

We turn next to an overview of the *nested sequential procedure* (or NSP) model, denoted by SP^0 . Further details and motivating examples are given in [14]. In certain respects, however, our presentation here will be more formal than that of [14]: in particular, our treatment of bound variables and issues of α -conversion will be more explicit, in order to provide a secure foundation for the detailed syntactic arguments that follow.

The ideas behind this model have a complex history. The general idea of sequential computation via nested oracle calls was the driving force behind Kleene’s later papers (e.g. [9]), although the concept did not receive a particularly transparent or definitive formulation there. Many of the essential ideas of NSPs can be found in early work of Sazonov [19], in which a notion of *Turing machine with oracles* was used to characterize the ‘sequentially computable’ elements of the Scott model. NSPs as we study them here were first explicitly introduced in work on game semantics for PCF—both by Abramsky, Jagadeesan and Malacaria [1] (under the name of *evaluation trees*) and by Hyland and Ong [7] (under the name of *canonical forms*). In these papers, NSPs played only an ancillary role; however, it was shown by Amadio and Curien [3] how (under the name of PCF *Böhm trees*) they could be made into a model of PCF in their own right. Similar ideas were employed again by Sazonov [20] to give a standalone characterization of the class of sequentially computable functionals. More recently, Normann and Sazonov [17] gave an explicit construction of the NSP model in a somewhat more semantic spirit than [3], using the name *sequential procedures*. As in [14], we here add the epithet ‘nested’ to emphasize the contrast with other flavours of sequential computation.¹

As in [14], our NSPs are generated by means of the following infinitary grammar, interpreted coinductively:

$$\text{Procedures: } p, q ::= \lambda x_0 \cdots x_{r-1}. e$$

¹A major theme of [14] is that NSPs serve equally well to capture the essence of PCF computation and that of Kleene’s S1–S9 computability; this is one reason for preferring a name that is not biased towards PCF.

Expressions: $d, e ::= \perp \mid n \mid \text{case } a \text{ of } (i \Rightarrow e_i \mid i \in \mathbb{N})$
Applications: $a ::= x q_0 \cdots q_{r-1}$

We will use vector notation to denote finite (possibly empty) lists of variables or procedures: \vec{x}, \vec{q} . Our convention will be that a list \vec{x} must be non-repetitive, though a list \vec{q} need not be. We may use t to range over *NSP terms* of any of the above three kinds. A procedure $\lambda \vec{x}. \perp$ will often be abbreviated to \perp .

For the most part, we will be working with terms modulo (infinitary) α -equivalence, and most of the concepts we introduce will be stable under renamings of bound variables. Thus, a statement $t = t'$, appearing without qualification, will mean that t, t' are α -equivalence (although we will sometimes write $=$ as $=_\alpha$ if we particularly wish to emphasize this). When we wish to work with terms on the nose rather than up to $=_\alpha$, we shall refer to them as *concrete* terms.

If each variable is assigned a simple type over \mathbb{N} , then we may restrict our attention to *well-typed* terms. Informally, a term will be well-typed unless a typing violation occurs at some specific point within its syntax tree. The typing discipline will mostly play only a background role in the present paper, but for the sake of completeness we note the rules here. Specifically, a term t is well-typed if:

- for every application $x\vec{q}$ appearing within t , the type of x has the form $\vec{\sigma} \rightarrow \mathbb{N}$ where $\vec{\sigma}$ and \vec{q} are of the same length; and
- for each i , the procedure q_i has the form $\lambda \vec{x}_i. e_i$, where the variables \vec{x}_i have types $\vec{\tau}_i$ and $\sigma_i = \vec{\tau}_i \rightarrow \mathbb{N}$.

If Γ is any environment (i.e. a finite non-repetitive list of variables), we write $\Gamma \vdash e$ and $\Gamma \vdash a$ to mean that e, a respectively are well-typed with free variables in Γ . We also write $\Gamma \vdash p : \tau$ when p is well-typed in Γ and of the form $\lambda \vec{x}. e$, where the variables \vec{x} have types $\vec{\sigma}$ and $\tau = \vec{\sigma} \rightarrow \mathbb{N}$.

We shall often refer to variable environments that arise from combining several lists of variables, which may be represented by different notations, e.g. Γ, V, \vec{x} thing. Since such environments are required to be non-repetitive, we take it to be part of the content a typing judgement such as $\Gamma, V, \vec{x} \vdash t(: \tau)$ that the entire list Γ, V, \vec{x} is non-repetitive. However, the *order* of variables within an environment will typically be of much less concern to us (clearly our typing judgements $\Gamma \vdash T(: \tau)$ are robust under permutations of Γ), and we will sometimes abuse notation by identifying a finite *set* Z of variables with the list obtained from some arbitrary ordering of it.

It will also be convenient to place another condition on concrete well-typed terms (not imposed in [14]) in order to exclude *variable hiding*. Specifically, we shall insist that if $\Gamma \vdash t(: \tau)$ then no variable of Γ appears as a bound variable within t , nor are there any *nested* bindings within t of the same variable x . (Clearly any concrete term not satisfying this restriction is α -equivalent to one that does.) This will help to avoid confusion in the course of some delicate arguments in which the issue of identity of variables is paramount.

With these ideas in place, we may take $\text{SP}(\sigma)$ to be the set of well-typed procedures of type σ modulo $=\alpha$, and $\text{SP}^0(\sigma) \subseteq \text{SP}(\sigma)$ the subset constituted by the *closed* procedures (i.e. those that are well-typed in the empty environment).

As in [14], we shall need to work not only with NSPs themselves, but with a more general calculus of NSP *meta-terms* designed to accommodate the intermediate forms that arise in the course of computations:

$$\begin{aligned} \text{Meta-procedures: } P, Q, R &::= \lambda \vec{x}. E \\ \text{Meta-expressions: } D, E &::= \perp \mid n \mid \text{case } G \text{ of } (i \Rightarrow E_i \mid i \in \mathbb{N}) \\ \text{Ground meta-terms: } G &::= E \mid x \vec{Q} \mid P \vec{Q} \end{aligned}$$

Here again, \vec{x} and \vec{Q} denote finite lists. We shall use T to range over meta-terms of any of the above three kinds. (Unless otherwise stated, we use upper-case letters for general meta-terms and lowercase ones for terms.) Once again, we will normally work with meta-terms up to (infinitary) α -equivalence, but may work with concrete meta-terms when required.

There are also some evident typing rules for meta-terms, leading to typing judgements $\Gamma \vdash P : \sigma$, $\Gamma \vdash E$, $\Gamma \vdash G$ for meta-procedures, meta-expressions and ground meta-terms respectively. These typing rules are the obvious adaptation of those for terms (see [14, Section 6.1.1]); again, they will play only a background role in this paper. We furthermore require that well-typed concrete meta-terms are subject to the no-variable-hiding condition. We will sometimes write e.g. $\Gamma \vdash P$ to mean that P is a well-typed meta-procedure in environment Γ , if the type itself is of no particular concern to us.

There is an evident notion of simultaneous capture-avoiding *substitution* $T[\vec{x} \mapsto \vec{Q}]$ for well-typed concrete terms. Specifically, if $\Gamma, \vec{x} \vdash T(: \tau)$ and $\Gamma, \vec{y} \vdash Q_i$ for each i , where $\vec{x} = x_0^{\sigma_0}, \dots, x_{r-1}^{\sigma_{r-1}}$, then we will have $\Gamma, \vec{y} \vdash T[\vec{x} \mapsto \vec{Q}](: \tau)$. Note that this may entail renaming of bound variables both within T (in order to avoid capture of variables in \vec{y}) and in the Q_i (in order to maintain the no-hiding condition for variables in \vec{x} and those bound within T). The details of how this renaming is performed will not matter, provided that for each T, \vec{x}, \vec{Q} as above we have a determinate choice of a suitable concrete term $T[\vec{x} \mapsto \vec{Q}]$, so that multiple appearances of the same substitution will always yield the same result. We also note that substitution is clearly well-defined on α -equivalence classes. Finally, we will say a substitution $[\vec{x} \mapsto \vec{Q}]$ *covers* a set V of variables if V consists of precisely the variables \vec{x} .

As a mild extension of the concept of meta-term, we have an evident notion of a *meta-term context* $C[-]$: essentially a meta-term containing a ‘hole’ $-$, which may be of meta-procedure, meta-expression or ground meta-term type (and in the case of meta-procedures, will carry some type σ). Our convention is that a context $C[-]$ is permitted to contain only a single occurrence of the hole $-$. Multi-hole contexts $C[-_0, -_1, \dots]$ will occasionally be used, but again, each hole $-_i$ may appear only once.

By the *local variable environment* associated with a concrete meta-term context $\Gamma \vdash C[-]$, we shall mean the set X of variables x bound within $C[-]$ whose

scope includes the hole, so that the environment in force at the hole is Γ, X . (The no-variable-hiding convention ensures that X and indeed Γ, X is non-repetitive.) Although in principle local variable environments pertain to particular choices of concrete contexts, most of the concepts that we define using such environments will be easily seen to be invariant under renamings of bound variables.

Next, there is a concept of *evaluation* whereby any concrete meta-term $\Gamma \vdash T (: \sigma)$ evaluates to an ordinary concrete term $\Gamma \vdash \ll T \gg (: \sigma)$. To define this, the first step is to introduce a *basic reduction* relation \rightsquigarrow_b for concrete ground meta-terms, which we do by the following rules:

- (b1) $(\lambda \vec{x}.E)\vec{Q} \rightsquigarrow_b E[\vec{x} \mapsto \vec{Q}]$ (β -rule).
- (b2) $\text{case } \perp \text{ of } (i \Rightarrow E_i) \rightsquigarrow_b \perp$.
- (b3) $\text{case } n \text{ of } (i \Rightarrow E_i) \rightsquigarrow_b E_n$.
- (b4) $\text{case } (\text{case } G \text{ of } (i \Rightarrow E_i)) \text{ of } (j \Rightarrow F_j) \rightsquigarrow_b$
 $\text{case } G \text{ of } (i \Rightarrow \text{case } E_i \text{ of } (j \Rightarrow F_j))$.

Note that the β -rule applies even when \vec{x} is empty: e.g. $\lambda.2 \rightsquigarrow_b 2$.

From this, a *head reduction* relation \rightsquigarrow_h on concrete meta-terms is defined inductively:

- (h1) If $G \rightsquigarrow_b G'$ then $G \rightsquigarrow_h G'$.
- (h2) If $G \rightsquigarrow_h G'$ and G is not a **case** meta-term, then

$$\text{case } G \text{ of } (i \Rightarrow E_i) \rightsquigarrow_h \text{case } G' \text{ of } (i \Rightarrow E_i) .$$

- (h3) If $E \rightsquigarrow_h E'$ then $\lambda \vec{x}.E \rightsquigarrow_h \lambda \vec{x}.E'$.

Clearly, for any meta-term T , there is at most one T' with $T \rightsquigarrow_h T'$. We call a meta-term a *head normal form* if it cannot be further reduced using \rightsquigarrow_h . The possible shapes of head normal forms are \perp , n , **case** $y\vec{Q}$ **of** $(i \Rightarrow E_i)$ and $y\vec{Q}$, the first three optionally prefixed by $\lambda \vec{x}$ (where \vec{x} may contain y).

We now define the *general reduction* relation \rightsquigarrow inductively as follows:

- (g1) If $T \rightsquigarrow_h T'$ then $T \rightsquigarrow T'$.
- (g2) If $E \rightsquigarrow E'$ then $\lambda \vec{x}.E \rightsquigarrow \lambda \vec{x}.E'$.
- (g3) If $Q_j = Q'_j$ except at $j = k$ where $Q_k \rightsquigarrow Q'_k$, then

$$\begin{aligned} x\vec{Q} &\rightsquigarrow x\vec{Q}' , \\ \text{case } x\vec{Q} \text{ of } (i \Rightarrow E_i) &\rightsquigarrow \text{case } x\vec{Q}' \text{ of } (i \Rightarrow E_i) . \end{aligned}$$

- (g4) If $E_i = E'_i$ except at $i = k$ where $E_k \rightsquigarrow E'_k$, then

$$\text{case } x\vec{Q} \text{ of } (i \Rightarrow E_i) \rightsquigarrow \text{case } x\vec{Q} \text{ of } (i \Rightarrow E'_i) .$$

It is easy to check that this reduction system is sound with respect to the typing rules. We emphasize that the relation \rightsquigarrow is defined on concrete meta-terms, although it is clear that it also gives rise to a deterministic reduction relation on their α -classes. An important point to note is that the terms t are precisely the meta-terms in *normal form*, i.e. those that cannot be reduced using \rightsquigarrow . We write \rightsquigarrow^* for the reflexive-transitive closure of \rightsquigarrow .

The above reduction system captures the finitary aspects of evaluation. In general, however, since terms and meta-terms may be infinitely deep, evaluation must be seen as an infinite process. To account for this infinitary aspect, we use some familiar domain-theoretic ideas.

We write \sqsubseteq for the evident syntactic orderings on concrete meta-procedures and on ground meta-terms: thus, $T \sqsubseteq U$ iff T may be obtained from U by replacing zero or more subterms (possibly infinitely many) by \perp . It is easy to see that for each σ , the set of all concrete procedure terms of type σ forms a directed-complete partial order under \sqsubseteq .

By a *finite* (concrete) term t , we shall mean one generated by the following grammar, this time construed inductively:

$$\begin{aligned} \text{Procedures:} \quad p, q &::= \lambda x_0 \dots x_{r-1}. e \\ \text{Expressions:} \quad d, e &::= \perp \mid n \mid \text{case } a \text{ of } (0 \Rightarrow e_0 \mid \dots \mid r-1 \Rightarrow e_{r-1}) \\ \text{Applications:} \quad a &::= x q_0 \dots q_{r-1} \end{aligned}$$

We regard finite terms as ordinary NSP terms by identifying the conditional branching $(0 \Rightarrow e_0 \mid \dots \mid r-1 \Rightarrow e_{r-1})$ with

$$(0 \Rightarrow e_0 \mid \dots \mid r-1 \Rightarrow e_{r-1} \mid r \Rightarrow \perp \mid r+1 \Rightarrow \perp \mid \dots).$$

We may now explain how a general meta-term T *evaluates* to a term $\ll T \gg$. This will in general be an infinite process, but we can capture the value of T as the limit of the finite portions that become visible at finite stages in the reduction. To this end, for any concrete meta-term T we define

$$\Downarrow_{\text{fin}} T = \{t \text{ finite} \mid \exists T'. T \rightsquigarrow^* T' \wedge t \sqsubseteq T'\}.$$

It is not hard to check that for any meta-term T , the set $\Downarrow_{\text{fin}} T$ is directed with respect to \sqsubseteq (cf. [14, Proposition 6.1.2]). We may therefore define $\ll T \gg$, the *value* of T , to be the ordinary concrete term

$$\ll T \gg = \bigsqcup (\Downarrow_{\text{fin}} T).$$

Note in passing that the value $\ll G \gg$ of a ground meta-term G may be either an expression or an application. In either case, it is certainly a ground meta-term. It is also easy to see that $\ll \lambda \vec{x}. E \gg = \lambda \vec{x}. \ll E \gg$, and that if $T \rightsquigarrow^* T'$ then $\ll T \gg = \ll T' \gg$.

Whilst we have defined our evaluation operation $\ll - \gg$ for concrete meta-terms, it is clear that this induces a well-defined evaluation operation on their α -classes, and for the most part this is all that we shall need. We also note that

the syntactic ordering \sqsubseteq on concrete terms induces a partial order \sqsubseteq on their α -classes, and that each $\text{SP}(\sigma)$ and $\text{SP}^0(\sigma)$ is directed-complete with respect to this ordering.

In the present paper, an important role will be played by the tracking of variable occurrences (and sometimes other subterms) through the course of evaluation. By inspection of the above rules for \rightsquigarrow , it is easy to see that if $T \rightsquigarrow T'$, then for any occurrence of a (free or bound) variable x within T' , we can identify a unique occurrence of x within T from which it originates (we suppress the formal definition). The same therefore applies whenever $T \rightsquigarrow^* T'$. In this situation, we may say that the occurrence of x within T' is a *residual* of the one within T , or that the latter is the *origin* of the former. Note, however, that these relations are relative to a particular reduction path $T \rightsquigarrow^* T'$: there may be other paths from T to T' for which the origin-residual relation is different.

Likewise, for any occurrence of x within $\ll T \gg$, we may pick some finite $t \sqsubseteq \ll T \gg$ containing this occurrence, and some $T' \sqsupseteq t$ with $T \rightsquigarrow^* T'$; this allows us to identify a unique occurrence of x within T that originates the given occurrence in $\ll T \gg$. It is routine to check that this occurrence in T will be independent of the choice of t and T' and of the chosen reduction path $T \rightsquigarrow^* T'$; we therefore have a robust origin-residual relationship between variable occurrences in T and those in $\ll T \gg$.

A fundamental result for NSPs is the *evaluation theorem*, which says that the result of evaluating a meta-term is unaffected if we choose to evaluate certain subterms in advance:

Theorem 4 (Evaluation theorem) *If $C[-_0, -_1, \dots]$ is any meta-term context with countably many holes and $C[T_0, T_1, \dots]$ is well-formed, then*

$$\ll C[T_0, T_1, \dots] \gg = \ll C[\ll T_0 \gg, \ll T_1 \gg, \dots] \gg .$$

The proof of this is logically elementary but administratively complex: see [14, Section 6.1.2].

One further piece of machinery will be useful: the notion of *hereditary η -expansion*, which enables us to convert a variable x into a procedure term (written x^η). The definition is by recursion on the type of x : if $x : \sigma_0, \dots, \sigma_{r-1} \rightarrow \mathbb{N}$, then

$$x^\eta = \lambda z_0^{\sigma_0} \dots z_{r-1}^{\sigma_{r-1}} . \text{case } x z_0^\eta \dots z_{r-1}^\eta \text{ of } (i \Rightarrow i) .$$

In particular, if $x : \mathbb{N}$ then $x^\eta = \lambda . \text{case } x \text{ of } (i \Rightarrow i)$. The following useful properties of η -expansion are proved in [14, Section 6.1.3] (we assume the terms in question are well typed):

$$\begin{aligned} \ll x^\eta \vec{q} \gg &= \text{case } x \vec{q} \text{ of } (i \Rightarrow i) , \\ \ll \lambda \vec{y} . p \vec{y}^\eta \gg &= p . \end{aligned}$$

The sets $\text{SP}(\sigma)$ may now be made into a total applicative structure SP by defining

$$(\lambda x_0 \dots x_r . e) \cdot q = \lambda x_1 \dots x_r . \ll e[x_0 \mapsto q] \gg .$$

Clearly the sets $\mathbf{SP}^0(\sigma)$ are closed under this application operation, so we also obtain an applicative substructure \mathbf{SP}^0 of \mathbf{SP} . It is easy to check that application in \mathbf{SP} is monotone and continuous with respect to \sqsubseteq . It is also shown in [14, Section 6.1.3] that both \mathbf{SP} and \mathbf{SP}^0 are typed λ -algebras: that is, they admit a compositional interpretation of typed λ -terms that validates β -equality. (The relevant interpretation of pure λ -terms is in fact given by three of the clauses from the interpretation of \mathbf{PCF}^Ω as defined below.)

2.3 Interpretation of PCF in \mathbf{SP}^0

A central role will be played by certain procedures $Y_\sigma \in \mathbf{SP}^0((\sigma \rightarrow \sigma) \rightarrow \sigma)$ which we use to interpret the PCF constants Y_σ (the overloading of notation will do no harm in practice). If $\sigma = \sigma_0, \dots, \sigma_{r-1} \rightarrow \mathbb{N}$, we define $Y_\sigma = \lambda g^{\sigma \rightarrow \sigma}. F_\sigma[g]$, where $F_\sigma[g]$ is specified corecursively up to α -equivalence by:

$$F_\sigma[g] =_\alpha \lambda x_0^{\sigma_0} \dots x_{r-1}^{\sigma_{r-1}}. \text{case } g(F_\sigma[g]) \ x_0^\eta \dots x_{r-1}^\eta \text{ of } (i \Rightarrow i).$$

(A concrete representative of Y_σ satisfying the no-hiding condition will of course feature a different choice of bound variables x_0, \dots, x_{r-1} at each level.) We may now give the standard interpretation of \mathbf{PCF}^Ω in \mathbf{SP}^0 . To each \mathbf{PCF}^Ω term $\Gamma \vdash M : \sigma$ we associate a procedure-in-environment $\Gamma \vdash \llbracket M \rrbracket_\Gamma : \sigma$ inductively as follows:

$$\begin{aligned} \llbracket x^\sigma \rrbracket_\Gamma &= x^{\sigma\eta} \\ \llbracket \hat{n} \rrbracket_\Gamma &= \lambda.n \\ \llbracket \text{succ} \rrbracket_\Gamma &= \lambda x. \text{case } x \text{ of } (i \Rightarrow i + 1) \\ \llbracket \text{pre} \rrbracket_\Gamma &= \lambda x. \text{case } x \text{ of } (0 \Rightarrow 0 \mid i + 1 \Rightarrow i) \\ \llbracket \text{ifzero} \rrbracket_\Gamma &= \lambda xyz. \text{case } x \text{ of } (0 \Rightarrow \text{case } y \text{ of } (j \Rightarrow j) \\ &\quad \mid i + 1 \Rightarrow \text{case } z \text{ of } (j \Rightarrow j)) \\ \llbracket Y_\sigma \rrbracket_\Gamma &= Y_\sigma \\ \llbracket C_f \rrbracket_\Gamma &= \lambda x. \text{case } x \text{ of } (i \Rightarrow f(i)) \\ \llbracket \lambda x^\sigma. M \rrbracket_\Gamma &= \lambda x^\sigma. \llbracket M \rrbracket_{\Gamma, x^\sigma} \\ \llbracket MN \rrbracket_\Gamma &= \llbracket M \rrbracket_\Gamma \cdot \llbracket N \rrbracket_\Gamma \end{aligned}$$

(In the clause for C_f , we interpret $f(i)$ as \perp whenever $f(i)$ is undefined.)

As is shown in [14], this interpretation is *adequate*, in the sense that $M \rightsquigarrow^* \hat{n}$ iff $\llbracket M \rrbracket = \lambda.n$, and *universal*, in the sense that every element of $\mathbf{SP}^0(\sigma)$ is the denotation of some closed $M : \sigma$ in \mathbf{PCF}^Ω . It follows from these facts that the structure \mathbf{SF} is a quotient of \mathbf{SP}^0 , and indeed is its *extensional collapse*; we shall write \approx for the equivalence relation on \mathbf{SP}^0 induced by the quotient map. It is also routine to check that the canonical interpretation of \mathbf{PCF}^Ω in \mathbf{SF} factors through the above interpretation in \mathbf{SP}^0 via this quotient map.

Our proof of Theorem 2 will proceed via a detailed analysis of the model \mathbf{SP}^0 . Specifically, in Sections 3 and 4 we will show the following:

Theorem 5 *For any $k \geq 1$, the elements $\llbracket Y_{k+1} \rrbracket$ and $\llbracket Y_{0 \rightarrow (k+1)} \rrbracket$ in SP^0 are not PCF_k^Ω -definable.*

In Section 5 we will go on to show that no $Z \approx \llbracket Y_{0 \rightarrow (k+1)} \rrbracket$ can be PCF_k^Ω -definable. This already shows that the languages PCF_k form a strict hierarchy, but to complete the picture, we resort to a more refined version of our methods in Section 6 to show the same for Y_{k+1} . This establishes Theorem 2.

Finally, we note that each set $\mathsf{SF}(\sigma)$ naturally carries an *observational ordering*, namely the partial order \preceq given by

$$x \preceq x' \quad \text{iff} \quad \forall f \in \mathsf{SF}(\sigma \rightarrow \mathbb{N}), n \in \mathbb{N}. (f \cdot x = n \Rightarrow f \cdot x' = n).$$

Clearly, the application operations \cdot are monotone with respect to \preceq . Moreover, it follows from an inequational version of the context lemma that the observational ordering on $\mathsf{SF}(\sigma_0, \dots, \sigma_{r-1} \rightarrow \mathbb{N})$ coincides with the pointwise ordering induced by the usual partial order on \mathbb{N}_\perp . We shall also use the symbol \preceq for the preorder on each $\mathsf{SP}^0(\sigma)$ induced by \preceq on SF ; clearly this coincides with the *observational preorder* on $\mathsf{SP}^0(\sigma)$ defined by

$$q \preceq q' \quad \text{iff} \quad \forall p \in \mathsf{SP}^0(\sigma \rightarrow \mathbb{N}), n \in \mathbb{N}. (p \cdot q = \lambda.n \Rightarrow p \cdot q' = \lambda.n).$$

Note too that $q \approx q'$ iff $q \preceq q' \preceq q$. We shall also extend the use of the notations \approx, \preceq in a natural way open terms (in the same environment) and indeed to meta-terms: for example, we may write $\vec{x} \vdash P \preceq P'$ to mean $\ll \lambda \vec{x}. P \gg \preceq \ll \lambda \vec{x}. P' \gg$.

2.4 The embeddability hierarchy

The following result will play a crucial role in this paper:

Theorem 6 (Strictness of embeddability hierarchy) *In SF , no type $\overline{k+1}$ can be a pseudo-retract of any finite product $\Pi_i \sigma_i$ where each σ_i is of level $\leq k$. More formally, if z is a variable of type $\overline{k+1}$ and each x_i a variable of type σ_i , there cannot exist procedures*

$$z \vdash t_i : \sigma_i, \quad \vec{x} \vdash r : \overline{k+1}$$

such that $z \vdash r[\vec{x} \mapsto \vec{t}] \succeq z^\eta$.

If in the above setting we had $z \vdash r[\vec{x} \mapsto \vec{t}] \approx z^\eta$, we would call $\overline{k+1}$ a *retract* of $\Pi_i \sigma_i$. In Appendix A we will show that the notions of retract and pseudo-retract actually coincide, since $z \vdash p \succeq z^\eta$ implies $z \vdash p \approx z^\eta$. However, this fact will not be needed for the main results of this paper.

In our statement of Theorem 6, we have referred informally to a product $\Pi_i \sigma_i$ which we have not precisely defined (although our formal statement gives everything that is officially necessary). One may readily make precise sense of this product notation within the *Karoubi envelope* $\mathbf{K}(\mathsf{SF})$ as studied in [14,

Chapter 4]: for instance, it is not hard to show that any finite product of level $\leq k$ types can be constructed as a retract of the pure type $\bar{k} + 1$.

The proof of Theorem 6 appears in [14, Section 7.7], but for self-containedness we reprise it here with some minor stylistic improvements.

PROOF By induction on k . For the case $k = 0$, we note that $\mathbf{N} \rightarrow \mathbf{N}$ cannot be a pseudo-retract of any \mathbf{N}^r , since (for example) the set of maximal elements in $\mathbf{SF}(\mathbf{N} \rightarrow \mathbf{N})$ is of larger cardinality than the set of all elements of $\mathbf{SF}(\mathbf{N})^r$. (Alternatively, one may note that $\mathbf{N} \rightarrow \mathbf{N}$ is not a *retract* of \mathbf{N}^r , since the former contains strictly ascending chains of length $r + 2$ while the latter does not; then use the method of Appendix A in the easy case $k = 1$ to show that any pseudo-retraction of the relevant type would be a retraction.)

Now assume the result for $k - 1$, and suppose for contradiction that $z \vdash t_i$ and $\vec{x} \vdash r$ exhibit $\bar{k} + 1$ as a pseudo-retract of $\Pi_i \sigma_i$ where each σ_i is of level $\leq k$. Let $v = \ll r[\vec{x} \mapsto \vec{t}] \gg$, so that $\ll v[z \mapsto u] \gg \succeq u$ for any $u \in \mathbf{SP}^0(k + 1)$. We first check that any v with this latter property must have the syntactic form $\lambda f^k. \text{case } zp \text{ of } (\dots)$ for some p of type \bar{k} . Indeed, it is clear that v does not have the form $\lambda f.n$ or $\lambda f.\perp$, and the only other alternative form is $\lambda f. \text{case } fp' \text{ of } (\dots)$. In that case, however, we would have

$$\ll v[z \mapsto \lambda w^k.0] \gg \cdot (\lambda y^{k-1}.\perp) = \perp,$$

contradicting $\ll v[z \mapsto \lambda w^k.0] \gg \cdot (\lambda y^{k-1}.\perp) \succeq (\lambda w.0)(\lambda y.\perp) = 0$.

We now focus on the subterm p in $v = \lambda f. \text{case } zp \text{ of } (\dots)$. The general direction of our argument will be to show that $\lambda f^k.p$ represents a function of type $\bar{k} \rightarrow \bar{k}$ that dominates the identity, and that moreover our construction of v as $\ll r[\vec{x} \mapsto \vec{t}] \gg$ can be used to split this into morphisms $\bar{k} \rightarrow \Pi_j \rho_j$ and $\Pi_j \rho_j \rightarrow \bar{k}$ where the ρ_j are of level $\leq k - 1$, contradicting the induction hypothesis. An apparent obstacle to this plan is that z as well as f may appear free in p ; however, it turns out that we still obtain all the properties we need if we specialize z here (somewhat arbitrarily) to $\lambda w.0$.

Specifically, we claim that $\ll p[f \mapsto q, z \mapsto \lambda w.0] \gg \succeq q$ for any $q \in \mathbf{SP}^0(k)$. For suppose that $q \cdot s = n \in \mathbf{N}$ whereas $\ll p[f \mapsto q, z \mapsto \lambda w.0] \gg \cdot s \neq n$ for some $s \in \mathbf{SP}^0(k - 1)$. Take $u = \lambda g. \text{case } gs \text{ of } (n \Rightarrow 0)$, so that $u \cdot q' = \perp$ whenever $q' \cdot s \neq n$. Then $u \leq \lambda w.0$ pointwise, so we have $\ll p[f \mapsto q, z \mapsto u] \gg \cdot s \neq n$ by the monotonicity of \mathbf{SF} (see the end of Section 2.3). By the definition of u , it follows that $\ll (zp)[f \mapsto q, z \mapsto u] \gg = \perp$, whence $\ll v[z \mapsto u] \gg \cdot q = \perp$, whereas $u \cdot q = 0$, contradicting $\ll v[z \mapsto u] \gg \succeq u$. We have thus shown that $\lambda f. \ll p[z \mapsto \lambda w.0] \gg \succeq id_k$.

Next, we show how to split the function represented by this procedure so as to go through some $\Pi_j \rho_j$ as above. Since $\ll r[\vec{x} \mapsto \vec{t}] \gg = \lambda f. \text{case } zp \text{ of } (\dots)$, we have that $r[\vec{x} \mapsto \vec{t}]$ reduces in finitely many steps to a head normal form $\lambda f. \text{case } zP \text{ of } (\dots)$ where $\ll P \gg = p$. By working backward through this reduction sequence, we may locate the ancestor within $r[\vec{x} \mapsto \vec{t}]$ of this head occurrence of z . Since z does not appear free in r , this occurs within some t_i , and clearly it must appear as the head of some subterm $\text{case } zP' \text{ of } (\dots)$ where

P is a substitution instance of P' .² Now since t_i has type σ_i of level $\leq k$, and $z : \bar{k} + 1$ is its only free variable, it is easy to see that all *bound* variables within t_i have pure types of level $< k$. Let x'_0, x'_1, \dots denote the finitely many bound variables that are in scope at the relevant occurrence of zP' , and suppose each x'_j has type ρ_j of level $< k$. By considering the form of the head reduction sequence $r[\vec{x} \mapsto \vec{t}] \rightsquigarrow_h^* \lambda f. \text{case } zP \text{ of } (\dots)$, we now see that P has the form $P'[\vec{x}' \mapsto \vec{T}']$ where each $T_j : \rho_j$ contains at most f and z free.

Writing $*$ for the substitution $[z \mapsto \lambda w.0]$, define procedures

$$f^k \vdash t'_j = \ll T_j^* \gg : \rho_j, \quad \vec{h} \vdash r' = \ll P'^* \gg : \bar{k}.$$

Then $\ll r'[\vec{x}' \mapsto \vec{t}'] \gg$ coincides with the term $\ll \lambda f. P^* \gg = \lambda f. \ll p^* \gg$, which dominates the identity as shown above. Thus \bar{k} is a pseudo-retract of $\Pi_j \rho_j$, which contradicts the induction hypothesis. So $\bar{k} + 1$ is not a pseudo-retract of $\Pi_i \sigma_i$ after all, and the proof is complete. \square

As an aside, we remark that for several extensions of PCF studied in the literature, the situation is completely different, in that the corresponding fully abstract and universal models possess a *universal* simple type v of which all simple types are retracts. It follows easily in these cases that one can indeed bound the type levels of recursion operators without loss of expressivity. For example:

- In the language $\text{PCF} + \text{por} + \text{exists}$ considered by Plotkin [18], the type $\mathbb{N} \rightarrow \mathbb{N}$ is universal, and the proof of this shows that every program in this language is observationally equivalent to one in $\text{PCF}_1 + \text{por} + \text{exists}$. (This latter fact was already noted in [18].)
- In the $\text{PCF} + \text{catch}$ (a slight strengthening of Cartwright and Felleisen's language SPCF [5]), the type $\mathbb{N} \rightarrow \mathbb{N}$ is again universal, and again the sublanguage $\text{PCF}_1 + \text{catch}$ has the same expressive power.
- In the language $\text{PCF} + H$ of Longley [11], the type $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$ is universal, but even here, all constants Y_σ with $\text{lv}(\sigma) > 1$ are dispensable.

Further details of each the above scenarios may be found in [14]. These facts may help to give some insight into why a cheap proof of our present results in the setting of pure PCF should not be expected.

2.5 Other sublanguages of PCF

Our main results will in effect present a hierarchy of sublanguages of PCF, with PCF_0 and PCF_1 as the first two rungs. However, there are also other sublanguages of interest that are either weaker than or incomparable with PCF_1 . We here offer a brief summary of some known results in order to situate our theorems within the broader picture.

²The reader wishing to see a more formal justification for this step may consult the proof of Lemma 19(i) below.

That PCF_1 surpasses PCF_0 in power is true but uninteresting, since the latter is an extremely weak language that cannot even define addition. More representative is that PCF_1 is strictly stronger than the language $\text{T}_0 + \text{min}$, where T_0 (a fragment of Gödel's System T) is the λ -calculus with constants

$$\hat{0} : \mathbb{N}, \quad \text{succ} : \mathbb{N} \rightarrow \mathbb{N}, \quad \text{rec}_0 : \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow \mathbb{N}).$$

Here rec_0 is the standard operator for primitive recursion, and min is the classical minimization operator of type $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$. On the one hand, it is an easy exercise to define both rec_0 and min in PCF_1 ; on the other hand, Berger [4] shows that the functional $F : (\mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$ recursively defined by

$$F \ h \ n = h \ n \ (F \ h \ (n + 1)),$$

whilst readily expressible in PCF_1 , is not definable in $\text{T}_0 + \text{min}$. This functional and its higher-type analogues will play a crucial role in Section 5 of the present paper.

This situation is revisited in [14] from the perspective of substructures of SP^0 : it is shown that $\text{T}_0 + \text{min}$, and indeed the whole of $\text{T} + \text{min}$, can be modelled within the substructure of *left-well-founded* procedures, but that the above functional F is not representable by any such procedure; thus F is not definable in $\text{T} + \text{min}$. (The reader may wish to study these results and proofs before proceeding further, as they provide a simpler instance of the basic method that we shall use in this paper.) We may make explicit a connection with programming language constructs here: for any types σ, τ one may consider an *iteration* operator $\text{iter}_{\sigma\tau} : (\sigma \rightarrow (\sigma + \tau)) \rightarrow (\sigma \rightarrow \tau)$ which embodies the behaviour of a *while* construct for imperative-style loops with state σ and exit type τ .³ It is an easy exercise to show that T_0 extended with all iterators $\text{iter}_{\sigma\tau}$ is powerful enough to define both minimization and all the System T recursors. Nonetheless, the procedures for $\text{iter}_{\sigma\tau}$ are easily seen to be left-well-founded, so that even this language cannot define the above functional F . We thus have a second-order functional that is definable by general recursion (and indeed in PCF_1) but not by iteration, even in its higher-type manifestations.

At third order, there are even ‘hereditarily total’ functionals definable in PCF_1 but not by higher-type iterators, one example being the well-known *bar recursion* operator (see [13]).

Even weaker than $\text{T}_0 + \text{min}$ is the language of (*strict*) *Kleene primitive recursion plus minimization*, denoted by Klex^{min} in [14]. This is in fact equivalent in power to a language with (a strict version of) $\text{iter}_{\mathbb{N}\mathbb{N}}$. It is also shown in [14] that its computational power corresponds precisely to that of computable *left-bounded* procedures; this is used to show that rec_0 is not computable in Klex^{min} and to obtain other non-definability results. It seems reasonable to regard left-bounded procedures as embodying the weakest higher-order computability notion of natural interest that is still Turing complete.

³The sum type $\sigma + \tau$ is not officially part of our system, but can (for any given σ, τ) be represented as a retract of some existing simple type.

3 Sequential procedures for PCF_k^Ω terms

For the remainder of the paper, we take k to be some fixed natural number greater than 0.

In this section we give an direct inductive characterization of the PCF_k^Ω -denotable elements of SP by making explicit how our interpretation works for terms of PCF_k^Ω . The first point to observe is that we may restrict attention to PCF_k^Ω terms in *long $\beta\eta$ -normal form*: that is, terms in β -normal form in which every variable or constant z of type $\sigma_0, \dots, \sigma_{r-1} \rightarrow \mathbb{N}$ is fully applied (i.e. appears at the head of a subterm $zN_0 \dots N_{r-1}$ of type \mathbb{N}). Moreover, an inductive characterization of the class of such terms is easily given:

Proposition 7 (i) *A procedure $\Gamma \vdash p : \sigma$ is denotable by a PCF_k^Ω term $\Gamma \vdash M : \sigma$ iff it is denotable by one in long $\beta\eta$ -normal form.*

(ii) *The class of long $\beta\eta$ -normal forms of PCF_k^Ω is inductively generated by the following clauses:*

1. *If $\Gamma \vdash N_i : \sigma_i$ is a normal form for each $i < r$ and $x^{\sigma_0, \dots, \sigma_{r-1} \rightarrow \mathbb{N}} \in \Gamma$, then $\Gamma \vdash xN_0 \dots N_{r-1} : \mathbb{N}$ is a normal form (note that r may be 0 here).*
2. *If $\Gamma, x^\sigma \vdash M : \tau$ is a normal form then so is $\Gamma \vdash \lambda x.M : \sigma \rightarrow \tau$.*
3. *The numeric literals $\Gamma \vdash \hat{n} : \mathbb{N}$ are normal forms.*
4. *If $\Gamma \vdash M : \mathbb{N}$ is a normal form then so are $\Gamma \vdash \text{suc } M : \mathbb{N}$, $\Gamma \vdash \text{pre } M : \mathbb{N}$ and $\Gamma \vdash C_f M : \mathbb{N}$ for any $f : \mathbb{N} \rightarrow \mathbb{N}$.*
5. *If $\Gamma \vdash M : \mathbb{N}$, $\Gamma \vdash N : \mathbb{N}$ and $\Gamma \vdash P : \mathbb{N}$ are normal forms, then so is $\Gamma \vdash \text{ifzero } M N P : \mathbb{N}$.*
6. *If $\sigma = \sigma_0, \dots, \sigma_{r-1} \rightarrow \mathbb{N}$ is of level $\leq k$ and $\Gamma \vdash M : \sigma \rightarrow \sigma$ and $\Gamma \vdash N_i : \sigma_i$ are normal forms, then $\Gamma \vdash Y_\sigma M N_0 \dots N_{r-1} : \mathbb{N}$ is a normal form.*

PROOF (i) It is a well-known property of simply typed λ -calculi that every term M is $\beta\eta$ -equivalent to one in long $\beta\eta$ -normal form: indeed, we may first compute the β -normal form of M and then repeatedly apply η -expansions to any subterms that are not already fully applied. Moreover, it is shown in [14, Theorem 6.1.18] that SP is a $\lambda\eta$ -algebra, so that if $\Gamma \vdash M \simeq_{\beta\eta} M'$ then $\llbracket M \rrbracket_\Gamma = \llbracket M' \rrbracket_\Gamma$ in SP . This establishes the claim.

(ii) This is clear from the fact that no application may be headed by a λ -abstraction and that all occurrences of variables and constants must be fully applied. \square

It follows that the class of PCF_k^Ω -denotable procedures may be generated inductively by a set of clauses that mirror the above formation rules for long $\beta\eta$ -normal PCF_k^Ω terms. We now consider each of these formation rules in turn in order to spell out the corresponding operation at the level of NSPs. In Section 4 we will show that these operations cannot give rise to *spinal* procedures, from which it will follow that no PCF_k^Ω -denotable procedure can be spinal.

For the first three formation rules, the effect on NSPs is easily described:

Proposition 8 (i) If $\Gamma \vdash xN_0 \dots N_{r-1} : \mathbb{N}$ in PCF^Ω , then $\llbracket xN_0 \dots N_{r-1} \rrbracket_\Gamma = \text{case } x \llbracket N_0 \rrbracket_\Gamma \dots \llbracket N_{r-1} \rrbracket_\Gamma \text{ of } (j \Rightarrow j)$.

(ii) If $\Gamma \vdash \lambda x.M : \sigma \rightarrow \tau$ in PCF^Ω , then $\llbracket \lambda x.M \rrbracket_\Gamma = \lambda x. \llbracket M \rrbracket_{\Gamma, x}$.

(iii) $\llbracket \hat{n} \rrbracket_\Gamma = \lambda.n$.

PROOF (i) Immediate from the definition of $\llbracket - \rrbracket$ and the fact that $\ll x^n \vec{q} \gg = \text{case } x \vec{q} \text{ of } (i \Rightarrow i)$ (see the end of Section 2.2).

(ii), (iii) are part of the definition of $\llbracket - \rrbracket$. \square

As regards the formation rules for *suc*, *pre*, *C_f* and *ifzero*, the situation is again fairly straightforward, although a little more machinery is needed:

Definition 9 (i) The set of rightward (occurrences of) numeral leaves within a term t is defined inductively by means of the following clauses:

1. A meta-term n is a rightward numeral leaf within itself.
2. Every rightward numeral leaf within e is also one within $\lambda \vec{x}.e$.
3. Every rightward numeral leaf in each e_i is also one in $\text{case } a \text{ of } (i \Rightarrow e_i)$.

(ii) If t is a term and e_i an expression for each i , let $t[i \mapsto e_i]$ denote the result of replacing each rightward leaf occurrence i in t by the corresponding e_i .

Lemma 10 $\ll \text{case } d \text{ of } (i \Rightarrow e_i) \gg = d[i \mapsto e_i]$ for any expressions d, e_i .

PROOF Define a ‘truncation’ operation $-^{(c)}$ on normal-form expressions for each $c \in \mathbb{N}$ as follows:

$$\begin{aligned} n^{(c)} &= n, & \perp^{(c)} &= \perp, \\ \text{case } g \text{ of } (i \Rightarrow e_i)^{(0)} &= \perp, \\ \text{case } g \text{ of } (i \Rightarrow e_i)^{(c+1)} &= \text{case } g \text{ of } (i \Rightarrow e_i^{(c)}). \end{aligned}$$

Then clearly $d = \bigsqcup_c d^{(c)}$ and $d[i \mapsto e_i] = \bigsqcup_c d^{(c)}[i \mapsto e_i]$. Moreover, we may show by induction on c that

$$\ll \text{case } d^{(c)} \text{ of } (i \Rightarrow e_i) \gg = d^{(c)}[i \mapsto e_i].$$

The case $c = 0$ is trivial since $d^{(0)}$ can only have the form n or \perp . For the induction step, the situation for $d = n, \perp$ is trivial, so let us suppose $d = \text{case } g \text{ of } (j \Rightarrow f_j)$. Then

$$\begin{aligned} &\ll \text{case } d^{(c+1)} \text{ of } (i \Rightarrow e_i) \gg \\ &= \ll \text{case } (\text{case } g \text{ of } (j \Rightarrow f_j^{(c)})) \text{ of } (i \Rightarrow e_i) \gg \\ &= \text{case } g \text{ of } (j \Rightarrow \ll \text{case } f_j^{(c)} \text{ of } (i \Rightarrow e_i) \gg) \\ &= \text{case } g \text{ of } (j \Rightarrow (f_j^{(c)}[i \mapsto e_i])) \text{ by induction hypothesis} \\ &= (\text{case } g \text{ of } (j \Rightarrow f_j^{(c)}))[i \mapsto e_i] \\ &= d^{(c+1)}[i \mapsto e_i]. \end{aligned}$$

Since $\ll - \gg$ is continuous, the proposition follows by taking the supremum over c . \square

From this lemma we may now read off the operations on NSPs that correspond to clauses 4 and 5 of Proposition 7(ii):

Proposition 11 (i) If $\Gamma \vdash M : \mathbb{N}$ in PCF^Ω , then $\llbracket C_f M \rrbracket_\Gamma = \llbracket M \rrbracket_\Gamma[i \mapsto f(i)]$ (understanding $f(i)$ to be \perp when $i \notin \text{dom } f$); similarly for *suc* and *pre*.

(ii) If $\Gamma \vdash M : \mathbb{N}$, $\Gamma \vdash N : \mathbb{N}$ and $\Gamma \vdash P : \mathbb{N}$, then $\llbracket \text{ifzero } M \ N \ P \rrbracket_\Gamma = \llbracket M \rrbracket_\Gamma[0 \mapsto d, i + 1 \mapsto e]$ where $\llbracket N \rrbracket_\Gamma = \lambda.d$ and $\llbracket P \rrbracket_\Gamma = \lambda.e$.

PROOF (i) The definition of $\llbracket - \rrbracket$ yields

$$\llbracket C_f M \rrbracket_\Gamma = \ll \text{case } \llbracket M \rrbracket_\Gamma \text{ of } (i \Rightarrow f(i)) \gg,$$

and by Lemma 10 this evaluates to $\llbracket M \rrbracket_\Gamma[i \mapsto f(i)]$. Likewise for *suc* and *pre*.

(ii) The definition of $\llbracket - \rrbracket$ yields

$$\llbracket \text{ifzero } M \ N \ P \rrbracket_\Gamma = \ll \text{case } \llbracket M \rrbracket_\Gamma \text{ of } (0 \Rightarrow d \mid i + 1 \Rightarrow e) \gg,$$

and by Lemma 10 this evaluates to $\llbracket M \rrbracket_\Gamma[0 \mapsto d, i + 1 \mapsto e]$. \square

It remains to consider the formation rule involving Y_σ . It will be convenient to regard the NSP for $YMN_0 \dots N_{r-1}$ as a result of plugging simpler NSPs together, in the sense indicated by the following definition. Here and later, we shall follow the convention that Greek capitals Γ, Δ denote arbitrary environments, while Roman capitals Z, X, V denote lists of variables of type level $\leq k$.

Definition 12 (Plugging) Suppose given the following data:

- a variable environment Γ ,
- a finite list Z of ‘plugging variables’ z of level $\leq k$, disjoint from Γ ,
- a root expression $\Gamma, Z \vdash e$,
- a substitution ξ assigning to each variable $z^\sigma \in Z$, a procedure $\Gamma, Z \vdash \xi(z) : \sigma$.

In this situation, we define the (k) -plugging $\Pi_{\Gamma, Z}(e, \xi)$ (often abbreviated to $\Pi(e, \xi)$) to be the meta-term obtained from e by repeatedly expanding variables $z \in Z$ to $\xi(z)$. More formally, writing T° for the meta-term obtained from T by replacing each subterm $z\vec{Q}$ with $z \in Z$ by \perp , we may define, up to α -equivalence,

$$\begin{aligned} \Pi^0(e, \xi) &= e, \\ \Pi^{m+1}(e, \xi) &= \Pi^m(e, \xi)[z \mapsto \xi(z) \text{ for all } z \in Z], \\ \Pi(e, \xi) &= \bigsqcup_m \Pi^m(e, \xi)^\circ, \end{aligned}$$

where \bigsqcup denotes supremum with respect to the syntactic order on meta-terms.

It is easy to see that $\Pi_{\Gamma,Z}(e, \xi)$ is well-typed in environment Γ . Note that some renaming of bound variables will typically be necessary in order to realize $\Pi_{\Gamma,Z}(e, \xi)$ as a concrete term conforming to the no-variable-hiding condition; we will not need to fix on any one particular way of doing this.

The operation on NSPs corresponding to clause 6 of Proposition 7(ii) may now be described as follows:

Proposition 13 *Suppose that $\sigma = \sigma_0, \dots, \sigma_{r-1} \rightarrow \mathbb{N}$ is of level $\leq k$ and that $\Gamma \vdash Y_\sigma M N_0 \dots N_{r-1}$ in PCF_k^Ω , where $\llbracket M \rrbracket_\Gamma = \lambda z^\sigma x_0^{\sigma_0} \dots x_{r-1}^{\sigma_{r-1}}.e$. Then*

$$\llbracket Y_\sigma M N_0 \dots N_{r-1} \rrbracket_\Gamma = \lambda. \ll \Pi_{\Gamma,Z}(e, \xi) \gg$$

where $Z = z, x_0, \dots, x_{r-1}$, $\xi(z) = \lambda x_0 \dots x_{r-1}.e$, and $\xi(x_i) = \llbracket N_i \rrbracket_\Gamma$ for each i .

PROOF Write $Y_\sigma = \lambda g.F_\sigma[g]$ where $F_\sigma[g] = \lambda \vec{x}.\text{case } g(F_\sigma[g]) \vec{x}^\eta \text{ of } (i \Rightarrow i)$ as at the start of Section 2.3. Then clearly

$$\llbracket Y_\sigma M \rrbracket_\Gamma = F_\sigma[\llbracket M \rrbracket_\Gamma] = \lambda \vec{x}.\ll \Pi_{\Gamma,\vec{x},\{z\}}(e, \xi') \gg \text{ where } \xi'(z) = \lambda \vec{x}.e,$$

and the proposition follows easily. \square

Note that in this instance of plugging, the repeated substitutions are needed only for the sake of the term $\xi(z)$ which may contain z free—only a single substitution step is needed for the plugging variables x_i .

Combining Propositions 8, 11 and 13 with Proposition 7, the results of this section may be summarized as follows.

Theorem 14 *The class of PCF_k^Ω -denotable procedures-in-environment $\Gamma \vdash p$ is the class generated inductively by the following rules:*

1. If $\Gamma \vdash q_i$ is denotable for each $i < r$ and $x \in \Gamma$, then

$$\Gamma \vdash \text{case } x q_0 \dots q_{r-1} \text{ of } (j \Rightarrow j)$$

is denotable.

2. If $\Gamma, x \vdash p$ is denotable, then $\Gamma \vdash \lambda x.p$ is denotable.
3. Each $\Gamma \vdash \lambda.n$ is denotable.
4. If $\Gamma \vdash p$ is denotable and $f : \mathbb{N} \rightarrow \mathbb{N}$, then $\Gamma \vdash p[i \mapsto f(i)]$ is denotable. (The constructions for *suc* and *pre* are special cases of this).
5. If $\Gamma \vdash p$, $\Gamma \vdash \lambda.d$ and $\Gamma \vdash \lambda.e$ are denotable, then $\Gamma \vdash p[0 \mapsto d, i+1 \mapsto e]$ is denotable.
6. If $\Gamma \vdash \lambda z^\sigma x_0^{\sigma_0} \dots x_{r-1}^{\sigma_{r-1}}.e$ is denotable where $\sigma = \sigma_0, \dots, \sigma_{r-1} \rightarrow \mathbb{N}$ is of level $\leq k$, and $\Gamma \vdash q_i : \sigma_i$ is denotable for each $i < r$, then

$$\Gamma \vdash \lambda. \ll \Pi_{\Gamma,Z}(e, \xi) \gg$$

is denotable, where $Z = z, \vec{x}$, $\xi(z) = \lambda \vec{x}.e$, and $\xi(x_i) = q_i$ for each i .

4 PCF_k^Ω -denotable procedures are not spinal

In this section, we will introduce the crucial notion of a *spinal term*, and will show that the clauses of Theorem 14 are unable to generate spinal terms from non-spinal ones. Since the element $Y_{k+1} \in \text{SP}^0$ is easily seen to be spinal, this will show that this procedure is not PCF_k^Ω -denotable. We will also explain how the notion of spinal term, with one slight modification, provides what we will need in order to show in Section 5 that $Y_{0 \rightarrow (k+1)}$ is not PCF_k^Ω -definable in SF.

Slightly more precisely, writing $Y_{k+1} = \lambda g.F_{k+1}[g]$ as before, we will show that the procedure $F_{k+1}[g]$ cannot be generated by the above constructions. This will suffice to show that Y_{k+1} itself cannot be so generated, since the only means of generating non-nullary λ -abstractions is via clause 2 of Theorem 14. To this end, we will actually introduce the notion of a *g-spinal term*, where $g : (k+1) \rightarrow (k+1)$ is a free variable which we shall treat as fixed throughout the whole of the subsequent discussion.

To motivate the definition of *g-spinal term*, let us try to explain informally the crucial difference between Y_{k+1} and Y_k (say) that we are trying to capture. The most obvious difference between these procedures is that Y_{k+1} involves an infinite sequence of nested calls to a variable g of type level $k+2$, whereas Y_k does not. One's first thought might therefore be to try and show that no procedure involving an infinite nesting of this kind can be constructed using the means at our disposal corresponding to PCF_k^Ω terms.

As it stands, however, this is not the case. Suppose, for example, that $up_k : k \rightarrow k+1$ and $down_k : k+1 \rightarrow k$ are PCF_0 terms defining a standard retraction $k \triangleleft k+1$. Specifically, let us inductively define

$$\begin{aligned} up_0 &= \lambda x^0.\lambda z^0.x, & down_0 &= \lambda y^1.y\hat{0}, \\ up_{k+1} &= \lambda x^{k+1}.\lambda z^{k+1}.x(down_k z), & down_{k+1} &= \lambda y^{k+2}.\lambda w^k.y(up_k w). \end{aligned}$$

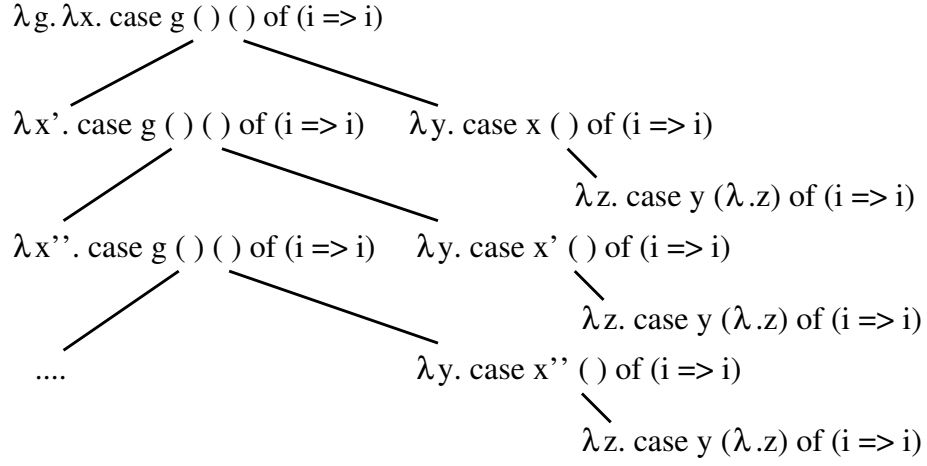
Now consider the PCF_k program

$$Z_{k+1} = \lambda g : (k+1) \rightarrow (k+1). up_k(Y_k(down_k \circ g \circ up_k)).$$

This is essentially just a representation of Y_k under our encoding of type k in type $k+1$. A simple calculation shows that the NSPs for Y_{k+1} and Z_{k+1} are superficially very similar in form, both involving an infinite sequence of nested calls to $g : (k+1) \rightarrow (k+1)$. (These NSPs are shown schematically in Figure 4 for the case $k=2$.) We will therefore need to identify some more subtle property of NSPs that differentiates between Y_{k+1} and Z_{k+1} .

The intuitive idea will be that in the NSP for Z_{k+1} , the full potency of g as a variable of type $k+1 \rightarrow k+1$ is not exploited, since both the input and output to g are ‘funnelled’ through the simpler type k . (Indeed, the force of Theorem 6 above is that the type k cannot fully represent the structure of the type $k+1$.) Broadly speaking, then, we shall define a *g-spinal term* to be one containing an infinite sequence of nested calls to g but with no essential ‘flattening’ of the arguments. It will then be the case that Y_{k+1} is *g-spinal*, but Z_{k+1} is not.

$Y_3 :$



$Z_3 :$

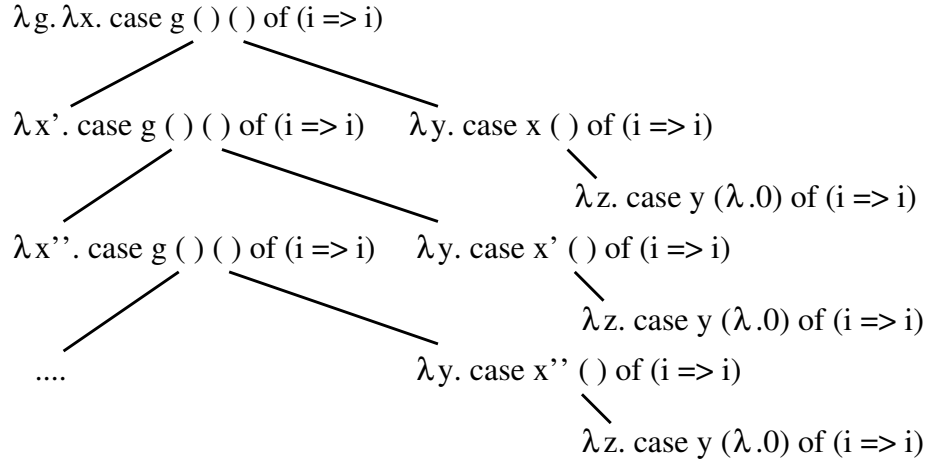


Figure 1: The NSPs for Y_3 and Z_3 . Here $\lambda.z$ abbreviates $\lambda. \text{case } z \text{ of } (i \Rightarrow i)$.

Some preliminaries are necessary before we can give the definition of g -spinal terms. We begin by identifying some properties that will be shared by all the terms-in-environment that we shall ever need to consider. The following ad hoc notions will be useful:

Definition 15 (i) An environment Γ is *regular* if Γ contains g but no other variables of type level $> k$.

(ii) A term-in-environment $\Gamma \vdash t$ is *regular* if Γ is regular and t is not a procedure of type level $> k + 1$.

Proposition 16 If $\Gamma \vdash t$ is regular, then all variables bound by a λ -abstraction within t are of level $\leq k$.

PROOF Suppose not, and suppose $\lambda\vec{x}.e$ is some outermost subterm of t with $\text{lv}(\vec{x}) > k$. Then $\lambda\vec{x}.e$ cannot be the whole of t , since t would then be a procedure of level $> k + 1$. Since t is a normal form, the subterm $\lambda\vec{x}.e$ (of level $> k + 1$) must therefore occur as an argument to some variable w of level $> k + 2$. But this is impossible, since Γ contains no such variables, nor can such a w be bound within t , since the relevant subterm $\lambda\vec{w}.d$ would then properly contain $\lambda\vec{x}.e$, contradicting the choice of the latter. \square

The above proposition suggests a generalization of Definition 15(ii) to meta-terms, which will sometimes be useful in the sequel:

Definition 17 A meta-term-in-environment $\Gamma \vdash T$ is *regular* if Γ is regular and all variables bound within T are of level $\leq k$ (whence T is not a meta-procedure of level $> k + 1$).

Let us now consider how the inductive clauses of Theorem 14 may be used to generate *regular* procedures-in-environment $\Gamma \vdash t$ is regular. An inspection of these clauses readily confirms that the procedures resulting from each of these clauses are regular only if the constituents from which they are constructed are regular. (There is a harmless exception for clause 6 in the case that not all the variables z, \vec{x} actually appear in e .) It follows that if $\Gamma \vdash t$ is regular and PCF_k^Ω -denotable, then there is an inductive generation of $\Gamma \vdash t$ via the clauses of Theorem 14 that consists entirely of regular procedures-in-environment; hence all λ -abstractions within these procedures will be of level $\leq k$. In particular, all of the above would apply to $g \vdash F_{k+1}[g]$ this procedure if it were PCF_k^Ω -denotable, since this is certainly regular.

We shall adopt the convention that any environment denoted by Γ will contain g as its only variable of level $> k$; recall that Roman letters such as V, X, Z always denote lists of variables of level $\leq k$ (which may also contribute to the environments we consider).

We are now ready to introduce the concept of a spinal term, generalizing the structure exhibited by the terms $F_{k+1}[g]$. In particular, we wish to allow the crucial applications of g to occur at positions other than the head of the enclosing abstractions $(\lambda x. \dots)$, and also to relax slightly the requirement that the second argument be a pure η -expanded variable.

Definition 18 (Spinal terms) (i) If x is a variable of type k and V a set of variables, a substitution $^\circ = [\vec{w} \mapsto \vec{r}]$ is called x, V -closed if the r_i contain no free variables, except that if $w_i \in V$ and $\text{lv}(w_i) < k$ then r_i may contain x free.

(ii) We coinductively declare a regular expression $\Gamma \vdash e$ to be (g) -head-spinal with respect to x, V iff e has the form

$$\text{case } g(\lambda x'. E[e']) o \text{ of } (\dots)$$

where $E[-]$ is any expression context, and

1. for some x, V -closed specialization $^\circ$ covering the free variables of o other than x , we have $o^\circ \succeq x^\eta$,
2. e' is head-spinal with respect to x', V' , where V' is the local variable environment for $E[-]$.

(That is, we take ‘ e is head-spinal w.r.t. x, V ’ to be the largest relation that satisfies the above equivalence.) In the above setting, we may also refer to the application $g(\lambda x'. E[e']) o$ as a head-spinal term.

(iii) We say a regular term $\Gamma \vdash t$ is (g) -spinal if it contains a head-spinal subexpression w.r.t. some x, V .

Whilst this definition makes use of local variable environments which in principle pertain to concrete terms, it is easily seen that the notion of spinal term is α -invariant.

Certain aspects of condition 1 in the definition deserve comment. For the purpose of showing the non-definability of Y_{k+1} as an element of \mathbf{SP}^0 , one could make do with the following simpler condition: for some *closed* specialization $^\circ$ of the free variables of o other than x , we have $\ll o^\circ \gg = x^\eta$. The looser condition adopted above is designed with the proof of non-definability in \mathbf{SF} in mind: we will see in Section 5 that every (simple) procedure representing a certain functional $\Phi \in \mathbf{SF}$ is spinal in the above sense. One might naturally expect to see the condition $o^\circ \approx x^\eta$ here, but it turns that the argument goes through most smoothly with \succeq in place of \approx . (In Appendix A we will see that $o^\circ \succeq x^\eta$ is actually equivalent to $o^\circ \approx x^\eta$, although this is not needed for our main proof.) The reason for requiring $^\circ$ to be x, V -closed is quite technical, but will emerge from the proof of Lemma 20.⁴

At this point, we digress briefly to explain the small modification of this machinery that we shall need for the results of Section 5. For reasons to be explained there, these results call for a setup in which the global variable g has the slightly different type $0 \rightarrow (k+1) \rightarrow (k+1)$. We may now vary the above definition by coinductively declaring e to be head-spinal w.r.t. x, V iff e has the form

$$\text{case } gb(\lambda x'. E[e']) o \text{ of } (\dots)$$

⁴It can be shown using Theorem 6 that if $o^\circ \succeq x^\eta$ where $^\circ$ is x, V -closed, then at least one x in $\ll o^\circ \gg$ must originate from o rather than from $^\circ$. We have not actually settled the question of whether there are procedures o such that $o^\circ \succeq x^\eta$ for some x, V -closed $^\circ$ but not for any closed $^\circ$; fortunately this is not necessary for the purpose of our proof.

where b is a procedure term of type 0 and conditions 1 and 2 above are also satisfied. Subject to this adjustment, all the results and proofs of the present section go through in this modified setting, with the extra argument b playing no active role. For the remainder of this section, we shall work with a global variable g of the simpler type $(k+1) \rightarrow (k+1)$, on the understanding that the extra arguments b can be inserted where needed to make formal sense of the material in the modified setting. We do not expect that any confusion will arise from this.

Clearly $g \vdash F_{k+1}[g]$ is spinal. The main result of this section will be that every PCF_k^Ω -denotable term $\Gamma \vdash t \in \mathbf{A}_k$ is non-spinal (Theorem 21). We shall establish this by induction on the generation of denotable terms as in Theorem 14, the only challenging case being the one for rule 6. Here we require some technical machinery to show that if the result of a plugging operation is spinal, then a spinal structure must already have been present in one of the components of the plugging: there is no way to ‘assemble’ a spinal structure from material in two or more non-spinal fragments.

The bulk of the proof will consist of some lemmas developing the machinery necessary for tackling rule 6. We start with some technical but essentially straightforward facts concerning evaluation and the tracking of subterms and variable substitutions.

Lemma 19 *Suppose that*

$$\Gamma \vdash \ll K[d] \gg =_\alpha K'[c]$$

where $K[-], K'[-]$ are concrete meta-term contexts with local environments \vec{v}, \vec{v}' respectively, and $\Gamma, \vec{v} \vdash d = \text{case } gpq \text{ of } (\dots), \Gamma, \vec{v}' \vdash c = \text{case } gp'q' \text{ of } (\dots)$ are concrete expressions. Suppose also that:

1. $\Gamma \vdash K[d]$ is regular;
2. in the evaluation above, the head g of c originates from that of d .

Then:

(i) There is a substitution $^\dagger = [\vec{v} \mapsto \vec{s}]$ of level $\leq k$, with $\Gamma, \vec{v}' \vdash \vec{s}$ regular, such that $\Gamma, \vec{v}' \vdash gp'q' =_\alpha \ll (gpq)^\dagger \gg$, whence $\ll d^\dagger \gg$ has the form $\text{case } gp'q' \text{ of } (\dots)$.

(ii) If furthermore c is head-spinal w.r.t. some x, V , then also $\ll d^\dagger \gg$ is head-spinal w.r.t. x, V .

(iii) If $K[-]$ contains no redexes $P\vec{Q}$ with P of type level $k+1$, then † is trivial for level k variables: that is, there is an injection ι mapping each level k variable $v_i \in \vec{v}$ to a variable $\iota(v_i) \in \vec{v}'$ such that $s_i = \iota(v_i)^\eta$.

Note that the environments \vec{v}, \vec{v}' , and hence the injection ι , will in general depend on the concrete choice of $K[d]$ and $K'[c]$. However, for the purpose of proving the theorem, it is clearly harmless to assume that $K'[c]$ is, on the nose, the concrete term obtained by evaluating $K[d]$. In this case, we will see from

the proof below that each $\iota(v_i)$ will be either v_i itself or a renaming of v_i arising from the evaluation.

PROOF (i) We formulate a suitable property of terms that is preserved under all individual reduction steps. Let $K[-], p, q$ and \vec{v} be fixed as above, and suppose that

$$K^0[\text{case } gP^0Q^0 \text{ of } (\dots)] \rightsquigarrow K^1[\text{case } gP^1Q^1 \text{ of } (\dots)]$$

via a single reduction step, where the g on the right originates from the one on the left, and moreover K^0, P^0, Q^0 enjoy the following properties (we write \vec{v}^0 for the local environment for $K^0[-]$):

1. $\Gamma \vdash K^0[\text{case } gP^0Q^0 \text{ of } (\dots)]$ is regular.
2. There exists a substitution $\dagger^0 = [\vec{v} \mapsto \vec{s}^0]$ (with $\Gamma, \vec{v}^0 \vdash \vec{s}^0$ regular) such that $\ll gP^0Q^0 \gg =_\alpha \ll (gpq)^{\dagger^0} \gg$.

We claim that K^1, P^1, Q^1 enjoy these same properties w.r.t. the local environment \vec{v}^1 for $K^1[-]$. For property 1, note that $K^1[\text{case } gP^1Q^1 \text{ of } (\dots)]$ cannot contain any bound variables of level $> k$ because $K^0[\text{case } gP^0Q^0 \text{ of } (\dots)]$ does not (see Proposition 16). For property 2, we define the required substitution $\dagger^1 = [\vec{v} \mapsto \vec{s}^1]$ by cases on the nature of the reduction step in question:

- If the subexpression $\text{case } gP^0Q^0 \text{ of } (\dots)$ is unaffected by the reduction (so that $P^0 = P^1$ and $Q^0 = Q^1$), or if the reduction is internal to P^0, Q^0 or to the rightward portion (\dots) , or if the reduction has the form

$$\begin{aligned} &\text{case } (\text{case } gP^0Q^0 \text{ of } (i \Rightarrow E_i^0)) \text{ of } (j \Rightarrow F_j) \rightsquigarrow \\ &\text{case } gP^0Q^0 \text{ of } (i \Rightarrow \text{case } E_i^0 \text{ of } (j \Rightarrow F_j)) \end{aligned}$$

then the conclusion is immediate, noting that $\vec{v}^1 = \vec{v}^0$ and taking $\dagger^1 = \dagger^0$.

- If the reduction is for a β -redex $(\lambda \vec{x}. E) \vec{R}$ where the indicated subexpression $\text{case } gP^0Q^0 \text{ of } (\dots)$ lies within some R_i , we may again take \dagger^1 to be \dagger^0 , with the obvious adjustments to compensate for any renaming of bound variables within R_i . In this case \vec{v}^1 may contain more variables than \vec{v}^0 , but we will still have that $\Gamma, \vec{v}^1 \vdash \vec{s}^1$.
- If the reduction is for a β -redex $(\lambda \vec{x}. E) \vec{R}$ where $\text{case } gP^0Q^0 \text{ of } (\dots)$ lies within E , then $P^1 = P^0[\vec{x} \mapsto \vec{R}]$ and similarly for Q^1 . In this case, the local environment \vec{v}^1 for $K^1[-]$ will be $\vec{v}^0 - \vec{x}$ (perhaps modulo renamings of the v_i^0), so that the conclusion follows if we take $\dagger^1 = [\vec{v} \mapsto \vec{s}^1]$ where $s_i^1 = \ll s_i^0[\vec{x} \mapsto \vec{R}] \gg$ for each i (modulo the same renamings).

Now in the situation of the lemma we will have some finite reduction sequence

$$K[\text{case } gpq \text{ of } (\dots)] \rightsquigarrow^* K''[\text{case } gP'Q' \text{ of } (\dots)],$$

where $\ll P' \gg = p'$, $\ll Q' \gg = q'$, and all subterms of $K''[-]$ containing the hole are in head normal form. Thus $\ll K''[-] \gg = K'[-]$, and there is a finite

normal-form context $t[-] \sqsubseteq K''[-]$ containing the hole in $K''[-]$ such that $t[-] \sqsubseteq K'[-]$. Since K, p, q themselves trivially satisfy the above invariants (taking $\dagger = [\vec{v} \mapsto \vec{v}^\eta]$), we may infer by iterating the argument above that K'', P', Q' also satisfy these invariants with respect to some $\dagger = [\vec{v} \mapsto \vec{s}]$ with $\Gamma, \vec{v}' \mapsto \vec{s}$ regular. (The environment Γ, \vec{v}' is correct here, as $K'[-], K''[-]$ have the same local environment.) We now have $gp'q' = \ll gP'Q' \gg =_\alpha \ll (gpq)^\dagger \gg$; note too that the \vec{v} are of level $\leq k$ because $K[d]$ is regular. It also follows immediately that $\ll d^\dagger \gg$ has the stated form.

(ii) If c is head-spinal w.r.t. x, V , then we see from Definition 18 that $gp'q'$ and hence $d = \text{case } gp'q' \text{ of } (\dots)$ are head-spinal w.r.t. x, V .

(iii) From the proof of (i), we see that in the reduction of $K[\text{case } gpq \text{ of } (\dots)]$ to $K''[\text{case } gP'Q' \text{ of } (\dots)]$, any $v_i \in \vec{v}$ can be tracked through the local environments for the intermediate contexts $K^0[-], K^1[-], \dots$ until (if ever) it is an eigenvariable for a β -reduction. For those v_i that never serve as an eigenvariable, it is clear from the construction that v_i gives rise to some variable $\iota(v_i) \in \vec{v}'$ (either v_i itself or a renaming thereof), and that $s_i = v_i^\dagger = \iota(v_i)^\eta$. We wish to show that all $v_i \in \vec{v}$ of level k are in this category.

By hypothesis, the meta-procedure P whose leading λ binds v_i does not occur in operator position; nor can it occur as an argument to another λ -abstraction within $K[-]$, since this would require a bound variable of level $\geq k+1$. It must therefore occur as a level $k+1$ argument to g , so that we have a subterm $g(\lambda v_i. E[-]) \dots$. But this form of subterms is stable under reductions, since g is a global variable; it follows easily that this subterm has a residue $g(\lambda v'_i. E'[-]) \dots$ in each of the intermediate reducts (where v'_i is either v_i or a renaming thereof), and thus that v_i and renamings thereof are never the eigenvariable of a β -reduction. \square

Thus, in the setting of the above lemma, the subterm d can be specialized and evaluated to yield a head-spinal term via the substitution $[\vec{v} \mapsto \vec{s}]$. However, we wish to show more, namely that in this setting, d itself is already a spinal term, so that the \vec{s} make no essential contribution to the spinal structure. (This is what we will need in order to show that k -pluggings cannot manufacture spinal terms out of non-spinal ones.) This is shown by the next lemma, whose proof forms the most complex and demanding part of our entire argument. The main challenge will be to show that all the head-spinal occurrences of g in $\ll d[\vec{v} \mapsto \vec{s}] \gg$ originate from d rather than from \vec{s} . The reader is advised that great care is needed regarding which variables can appear free where, and for this reason we shall make a habit of explicitly recording the variable environment for practically every term or meta-term that we mention.

Lemma 20 *Suppose we have regular terms*

$$\Gamma, \vec{v} \vdash d = \text{case } gpq \text{ of } (\dots), \quad \Gamma, \vec{v}' \vdash \vec{s}, \quad \text{lv}(\vec{v}), \text{lv}(\vec{v}') \leq k,$$

where $\Gamma, \vec{v}' \vdash \ll d[\vec{v} \mapsto \vec{s}] \gg$ is head-spinal with respect to some x, V . Then d itself is spinal.

PROOF We begin with some informal intuition. The term $\ll d[\vec{v} \mapsto \vec{s}] \gg$, being head-spinal, will be of the form

$$\Gamma, \vec{v}' \vdash t = \text{case } g(\lambda x'. E[\text{case } gF'o' \text{ of } (\dots)]) o \text{ of } (\dots),$$

where $o^\circ \succeq x'^\eta$ for some o (and likewise for o and x). Here the $\lambda x'$ clearly originates from the leading λ of p within d rather than from \vec{s} . Suppose, however, that the second spinal occurrence of g in t originated from some s_i rather than from d . In order to form the application of this g to o' , the whole content of x'^η would in effect need to be ‘passed in’ to s_i when d and \vec{s} are combined. But this is impossible, since the arguments to s_i are of level $< k$, so by Theorem 6 we cannot funnel the whole of x'^η through them: that is, the interface between d and \vec{s} is too narrow for the necessary interaction to occur. (The situation is made slightly more complex by the fact that some components of o might also involve x' , but the same idea applies.) It follows that the second spinal g in t originates from d . By iterating this argument, we can deduce that all the spinal occurrences of g , and indeed the entire spinal structure, comes from d .

We now proceed to the formal proof. By renaming variables if necessary, we may assume for clarity that the same variable is never bound in two places within the entire list of terms d, \vec{s} , and that all bound variables within d, \vec{s} are distinct from those of \vec{v} and \vec{v}' .

Let $\dagger = [\vec{v} \mapsto \vec{s}]$, and consider the subterm $p = \lambda x'^k.e$ within d , where $\Gamma, \vec{v}, x' \vdash e$ is regular. Since $\ll d^\dagger \gg$ is head-spinal, $\ll e^\dagger \gg$ will be some spinal term $\Gamma, x', \vec{v}' \vdash E[c]$, where $\Gamma, x', \vec{v}', \vec{y}' \vdash c = \text{case } gF'o' \text{ of } (\dots)$ is head-spinal with respect to x' and \vec{y}' . (Here \vec{y}' denotes the local environment for $E[-]$, so that $\Gamma, x', \vec{v}', \vec{y}'$ contains no repetitions.) We will first show that the head g of c comes from e rather than from \dagger ; we will later show that the same argument can be repeated for lower spinal occurrences of g .

Claim 1: In the evaluation $\ll e^\dagger \gg = E[c]$, the head g of c originates from e .

Proof of Claim 1: Suppose for contradiction that the head g of c originates from some substituted occurrence of an s_i within e^\dagger , say as indicated by $e^\dagger = D[s_i]$ and $s_i = L[d']$, where $\Gamma, x', \vec{v}' \vdash D[-]$, $\Gamma, \vec{v}' \vdash s_i$, and $\Gamma, \vec{v}', \vec{z} \vdash d' = \text{case } gp'q' \text{ of } (\dots)$. (Here \vec{z} is the local variable environment for $L[-]$; note that \vec{z} is disjoint from Γ, x', \vec{v}' , but may well overlap with \vec{y}' .) Then

$$\Gamma, x', \vec{v}' \vdash \ll e^\dagger \gg = \ll D[L[d']] \gg = E[c],$$

where the head g in d' is the origin of the head g in c . We will use this to show that a head-spinal term may be obtained from d' via a substitution of level $< k$; this will provide the bottleneck through which x'^η is unable to pass.

We first note that the above situation satisfies the conditions of Lemma 19, where we take the Γ, K, d, K', c of the lemma to be respectively (Γ, x', \vec{v}') , $D[L[-]]$, d', E, c . Condition 1 of the lemma holds because $\Gamma, \vec{v}, x' \vdash e$ and $\Gamma, \vec{v}' \vdash \vec{s}$ are clearly regular, so that e and hence e^\dagger involve no bound variables of level $> k$ (see Proposition 16); conditions 2 and 3 are immediate in the present setup.

We conclude that there is a substitution $[\vec{y} \mapsto \vec{t}]$ (called $[\vec{v} \mapsto \vec{s}]$ in the statement of Lemma 19) with \vec{y} the local environment for $D[L[-]]$ and $\Gamma, x', \vec{v}', \vec{y}' \vdash \vec{t}$ (recalling that \vec{y}' are the local variables for $E[-]$), such that $\ll d'[\vec{y} \mapsto \vec{t}] \gg$ is head-spinal and indeed of the form **case** $gF'o'$ **of** (\dots) with F', o' as above. Furthermore, the only β -redexes in e^\dagger are those arising from the substitution † , with some s_j of level k as operator. There are therefore no β -redexes in e^\dagger with an eigenvariable of level k , so by Lemma 19(iii), the substitution $[\vec{y} \mapsto \vec{t}]$ is trivial for variables of level k . Note also that \vec{y} (the environment for $D[L[-]]$) subsumes \vec{z} (the environment for $L[-]$); it is disjoint from Γ, x, \vec{v}' but may well overlap with \vec{y}' .

Let us now split the substitution $[\vec{y} \mapsto \vec{t}]$ as $[\vec{y}^+ \mapsto \vec{t}^+, \vec{y}^- \mapsto \vec{t}^-]$, where \vec{y}^+ consists of the variables in \vec{y} of level k , and \vec{y}^- consists of those of level $< k$. By Lemma 19(iii), the substitution for \vec{y}^+ is trivial: that is, there is a mapping associating with each $y_j \in \vec{y}^+$ a variable $\iota(y_j) \in \vec{y}'$ such that $t_j = \iota(y_j)^\eta$. (Indeed, the proof of Lemma 19(iii) shows that these variables are never specialized by a β -reduction in the course of evaluating $D[L[d']]$, though they may be renamed.)

Summarizing, we have that

$$\begin{aligned} \Gamma, x', \vec{v}', \vec{y}' &\vdash \ll d'[\vec{y} \mapsto \vec{t}] \gg = \text{case } gF'o' \text{ of } (\dots), \\ \Gamma, \vec{v}', \vec{z} &\vdash d' = \text{case } gp'q' \text{ of } (\dots), \\ \Gamma, x', \vec{v}', \vec{y}' &\vdash \vec{t}, \end{aligned}$$

where $[\vec{y} \mapsto \vec{t}]$ is trivial for level k variables, and $gF'o'$ is head-spinal w.r.t. x, \vec{y}' . From this we may read off that

$$\Gamma, x', \vec{v}', \vec{y}' \vdash \ll q'[\vec{y} \mapsto \vec{t}] \gg = o'.$$

We may henceforth regard q' as a term in environment $\Gamma, \vec{v}', \vec{y}$, since our conventions ensure that the variables of $\vec{y} - \vec{z}$ come from d rather than s_i and so do not appear bound in q' . harmlessly write $q'[\vec{y} \mapsto \vec{t}]$ even though the variables of $\vec{y} - \vec{z}$ do not appear in q' .

Since x' does not occur free in q' , each free occurrence of x' in o' above must originate from some $t_j \in \vec{t}$, which must furthermore have some type ρ_j of level $< k$, since if t_j had level k then we would have $t_j = \iota(y_j)^\eta$ which does not contain x' free. In fact, we may decompose the substitution $[\vec{y} \mapsto \vec{t}]$ as $[\vec{y}^+ \mapsto \iota(\vec{y}^+)^\eta]$ followed by $[\vec{y}^- \mapsto \vec{t}^-]$, since none of variables of \vec{y}^- appear free in the $\iota(y_j)^\eta$ for $y_j \in \vec{y}^+$. Setting $q'^* = \ll q[\vec{y}^+ \mapsto \iota(\vec{y}^+)^\eta] \gg$ (so that q'^* is just q' with the variables in \vec{y}^+ rewritten via ι), we therefore have $\ll q'^*[\vec{y}^- \mapsto \vec{t}^-] \gg = o'$. Thus:

$$\begin{aligned} \Gamma, \vec{v}', \iota(\vec{y}^+), \vec{y}^- &\vdash q'^* : \bar{k}, \\ \Gamma, x, \vec{v}', \vec{y}' &\vdash t_j : \rho_j \text{ for } t_j \in \vec{t}^-, \\ \Gamma, x, \vec{v}', \vec{y}' &\vdash \ll q'^*[\vec{y}^- \mapsto \vec{t}^-] \gg = o'. \end{aligned}$$

We may exploit this to manifest \bar{k} as a pseudo-retract of a level $< k$ product type, contradicting Theorem 6. Specifically, recalling that $gF'o'$ is head-spinal

w.r.t. x, \vec{y}' , take $\circ = [\vec{w} \mapsto \vec{r}]$ an x', \vec{y}' -closed substitution such that $o^\circ \succeq x'^\eta$ as in Definition 18; we may assume that \vec{w} is exactly $\Gamma, \vec{v}', \vec{y}'$. Reordering our variables, we may now write $x, \vec{w} \vdash \vec{t}^-$.

Next, let us split \circ into two independent parts: a substitution $[\vec{w}^+ \mapsto \vec{r}^+]$ covering the variables in $\Gamma, \vec{v}', \vec{y}'$ of level $\geq k$ (where $\vdash \vec{r}^+$), and $[\vec{w}^- \mapsto \vec{r}^-]$ covering those of level $< k$ (where $x' \vdash \vec{r}^-$). Set $q'^l = \ll q'^*[\vec{w}^+ \mapsto \vec{r}^+] \gg$, so that $\vec{u}^- \vdash q'^l$ where \vec{u}^- consists of the variables of $\Gamma, \vec{v}', \vec{y}'$ of level $< k$. The idea is that $\vec{u}^- \vdash q'^l$ may now serve as one half of a suitable pseudo-retraction. For the other half, let $[\vec{u}^- \mapsto \vec{a}^-]$ denote the effect of the substitution $[\vec{y}^- \mapsto \vec{t}^-]$ followed by $\circ = [\vec{w} \mapsto \vec{r}]$ (the order is important here as \vec{y}^- and \vec{w} may overlap). Since $\vec{u}^- \subseteq \vec{y}^- \cup \vec{w}$ and $x, \vec{w} \vdash \vec{t}^-$ and $x \vdash \vec{r}$, this substitution does indeed cover at least the variables of \vec{u}^- and we have $x \vdash \vec{a}^-$. We may now verify that $\vec{u}^- \vdash q'^l$ and $x \vdash \vec{a}^-$ constitute a pseudo-retraction as follows:

$$\begin{aligned} x' &\vdash \ll q'^l[\vec{u}^- \mapsto \vec{a}^-] \gg \\ &= \ll q'^*[\vec{w}^+ \mapsto \vec{r}^+][\vec{y}^- \mapsto \vec{t}^-][\vec{w} \mapsto \vec{r}] \gg \\ &= \ll (q'^*[\vec{y}^- \mapsto \vec{t}^-])[\vec{w} \mapsto \vec{r}] \gg \\ &= \ll o'^\circ \gg \succeq x'^\eta. \end{aligned}$$

As regards the second equation, the first substitution $[\vec{w}^+ \mapsto \vec{r}^+]$ may be safely omitted as \vec{w}^+ and \vec{y}^- are disjoint and the terms \vec{r}^+ do not contain any of the \vec{y}^- free. We therefore have \vec{k} as a pseudo-retract of a product of level $< k$ types. This contradicts Theorem 6, so the proof of Claim 1 is complete.

In order to continue this analysis to greater depth, let us now suppose that the originating occurrence of the head g in c is as indicated by $\Gamma, x', \vec{v} \vdash e = C[d']$, where $\Gamma, x', \vec{v}, \vec{v}'' \vdash d' = \text{case } gp'q' \text{ of } (\dots)$. (Here \vec{v}'' is the local environment for $C[-]$. The symbols d', p', q' are available for recycling now that the proof of Claim 1 is complete.) Then we have

$$\Gamma, x', \vec{v}' \vdash \ll e^\dagger \gg = \ll (\lambda \vec{v}. C[d'])\vec{s} \gg = E[c],$$

where c is head-spinal w.r.t. x', \vec{y}' , and the head g of d' is the origin of the head g of c . (Recall that \vec{y}' is the local environment for $E[-]$ and that $e = C[d']$ is regular, whence $\text{lv}(\vec{v}'') \leq k$.)

We claim that once again we are in the situation of Lemma 19, taking Γ, K, d, K', c of the lemma to be respectively (Γ, x', \vec{v}') , $(\lambda \vec{v}. C[-])\vec{s}$, d', E, c . Conditions 2 and 3 of the lemma are immediate in the present setup; for condition 1, we again note that $\Gamma, x', \vec{v}' \vdash C[d'] = e$ and $\Gamma, \vec{v}' \vdash \vec{s}$ are regular, so by Proposition 16 contain no bound variables of level $> k$; hence the same is true for $(\lambda \vec{v}. C[d'])\vec{s}$. Applying Lemma 19, we obtain a substitution $^\dagger = [\vec{v}^+ \mapsto \vec{s}^+]$ of level $\leq k$ (with $\vec{v}^+ = \vec{v}, \vec{v}''$), where $\Gamma, x', \vec{v}', \vec{v}^+ \vdash d'$ and $\Gamma, x', \vec{v}', \vec{y}' \vdash \vec{s}^+$ are regular, such that

$$\Gamma, x', \vec{v}', \vec{y}' \vdash \ll d'[\vec{v}^+ \mapsto \vec{s}^+] \gg$$

is head-spinal w.r.t. x', \vec{y}' , and indeed of the form **case** $gF'o'$ of (\dots) with F', o' as above. (We may in fact write just $\Gamma, x', \vec{v}^+ \vdash d'$, since by assumption the variables of \vec{v}' do not overlap with the free or bound variables of d so do not appear in d .) We may also read off that $\ll (p')^{\dagger'} \gg = F'$ and $\ll (q')^{\dagger'} \gg = o'$. As regards the substitution $\dagger' = [\vec{v}^+ \mapsto \vec{s}^+]$, it is clear that this extends $[\vec{v} \mapsto \vec{s}]$ since the evaluation of $(\lambda \vec{v}. C[d'])\vec{s}$ starts by β -reducing this term. Moreover, the argument of Lemma 19(iii) shows that \dagger' is trivial for any level k variables in \vec{v}'' , as $C[-]$ is in normal form.

We are now back precisely where we started, in the sense that d', \vec{v}^+, \vec{s}^+ themselves satisfy the hypotheses of Lemma 20, with (Γ, x') now playing the role of Γ and (\vec{v}', \vec{y}') that of \vec{v}' . Explicitly, we have regular terms

$$\Gamma, x, \vec{v}^+ \vdash d' = \text{case } gp'q' \text{ of } (\dots), \quad \Gamma, x, \vec{v}', \vec{y}' \vdash \vec{s}^+$$

(so that $\text{lv}(\vec{v}^+), \text{lv}(\vec{v}', \vec{y}') \leq k$) where $\Gamma, x, \vec{v}', \vec{y}' \vdash \ll d'[\vec{v}^+ \mapsto \vec{s}^+] \gg$ is head-spinal w.r.t. x', W . We can therefore iterate the above argument to obtain an infinite descending chain of subterms

$$\begin{array}{llll} \Gamma, \vec{v} \vdash & d = & \text{case } gpq \text{ of } (\dots), & p = \lambda x'. C[d'], \\ \Gamma, \vec{v}, x', \vec{v}'' \vdash & d' = & \text{case } gp'q' \text{ of } (\dots), & p' = \lambda x''. C'[d''], \\ \Gamma, \vec{v}, x', \vec{v}'', x'', \vec{v}''' \vdash & d'' = & \text{case } gp''q'' \text{ of } (\dots), & p'' = \lambda x'''. C''[d'''], \\ & \dots & & \dots \end{array}$$

along with associated substitutions $\dagger, \dagger', \dagger'', \dots$ applicable to d, d', d'', \dots respectively, such that the terms $\ll p^\dagger \gg, \ll (p')^{\dagger'} \gg, \ll (p'')^{\dagger''} \gg, \dots$ coincide with the successive procedure subterms F, F', \dots from the spine of the original t , and likewise $\ll q^\dagger \gg, \ll (q')^{\dagger'} \gg, \ll (q'')^{\dagger''} \gg, \dots$ coincide with o, o', \dots .

We cannot quite conclude that d is head-spinal, because the critical x in q^\dagger might originate not from q but from a level k term in \vec{s} (for example). However, we can show that this problem does not arise for q', q'', \dots , essentially because x', x'', \dots are bound locally within p . We will in fact show that d' is head-spinal w.r.t. x', \vec{v}'' , where \vec{v}'' is the local environment for $C[-]$; this will imply that d is spinal. In the light of Definition 18, it will be sufficient to show that $(q')^{\circ'} \succeq x'^\eta$ for some x', \vec{v}'' -closed specialization \circ' covering the free variables of q' except x' (namely those of $\Gamma, \vec{v}, \vec{v}''$); the same argument will then obviously apply also to q'', q''', \dots .

Recall that $\Gamma, \vec{v}, \vec{v}'', x' \vdash q'$ and $\Gamma, \vec{v}', \vec{y}', x' \vdash o'$. Since $gF'o'$ is head-spinal w.r.t. x, \vec{y}' , we may as before take $\circ = [\vec{w} \mapsto \vec{r}]$ x', \vec{y}' -closed such that $o'^\circ \succeq x'^\eta$, where $\vec{w} = \Gamma, \vec{v}', \vec{y}'$ and $x' \vdash \vec{r}$. Now define

$$\circ' = [\vec{v} \mapsto \vec{s}^\circ, \vec{v}'' \mapsto (\vec{s}'')^\circ, \vec{w}^\Gamma \mapsto \vec{r}^\Gamma],$$

(where we write \vec{s}^+ as \vec{s}, \vec{s}'' , and $\vec{w}^\Gamma \mapsto \vec{r}^\Gamma$ denotes the restriction of \circ to Γ). This covers the free variables of q' except x' , and we have $x' \vdash \vec{s}^\circ, (\vec{s}'')^\circ, \vec{r}^\Gamma$ because $\vec{w} \vdash \vec{s}, \vec{s}''$ and $x \vdash \vec{r}$. Moreover, we have

$$q'^{\circ'} = (q'[\vec{v}^+ \mapsto (\vec{s}^+)^\circ])[\vec{w}^\Gamma \mapsto \vec{r}^\Gamma] = (q'^{\dagger'})^\circ \approx o'^\circ \succeq x'^\eta$$

since $(\vec{s}^+)^{\circ}$ contains no free variables except x . To check that $^{\circ}$ is x', \vec{v}'' -closed, it remains to show that that u° may contain x' free only when $u \in \vec{v},''$ and u is of level $< k$. (Indeed, it is because of the possibility of x' occurring free in these terms that the machinery of x, V -closed substitutions is necessary at all.) The remaining cases are handled as follows:

- The terms \vec{s} exist in environment Γ, \vec{v}' , so do not involve x' or any of the variables of \vec{y}' . Since $^{\circ}$ is x', \vec{y}' -closed, it follows that the terms \vec{s}° do not involve x' .
- For any variables $v \in \vec{v}''$ of level k , we have $v^{\dagger} = v^{\eta}$ which contains no free variables of level $< k$, so that $(v^{\dagger})^{\circ}$ cannot involve x' .
- The \vec{r}^{Γ} cannot involve x' , since $^{\circ}$ is x', \vec{y}' -closed and Γ is disjoint from \vec{y}' .

This completes the verification that d is spinal. \square

From the above lemma we may immediately conclude, for example, that in the setting of Lemma 19 the term d is spinal.

We are now ready for the main result of this section:

Theorem 21 *Every PCF_k^{Ω} -denotable procedure $\Gamma \vdash p$ is non-spinal.*

PROOF In the light of Section 3, it will suffice to show that the clauses of Theorem 14 cannot generate spinal terms from non-spinal ones. For clauses 1–5 this is very easily seen. For clause 6, it will be sufficient to show that non-spinal terms are closed under k -plugging, and it is here that the machinery of Lemmas 19 and 20 comes into play.

Suppose that $t = \ll \Pi_{\Gamma, Z}(e, \xi) \gg$ where $\Gamma, Z \vdash e$ and $\Gamma, Z \vdash \xi(z)$ for each $z \in Z$. For later convenience, to each $z_i \in Z$ let us associate the procedure $\Gamma \vdash r_i = \ll \Pi_{\Gamma, Z}(\xi(z_i), \xi) \gg$; it is then clear from the definition of plugging and the evaluation theorem that $t = \ll e[\vec{z} \mapsto \vec{r}] \gg$ and that $r_i = \ll \xi(z_i)[\vec{z} \mapsto \vec{r}] \gg$ for each i .

It will be natural to frame our argument contrapositively. So suppose that t is spinal, i.e. t contains some head-spinal expression c at position $K[-]$. We shall focus on the head occurrence of g in c . Clearly this occurrence must originate from some normal-form term fragment $\Gamma, Z \vdash t_0$ involved in the above plugging (which may be either e or some $\xi(z)$). We will show that t_0 itself is spinal.

Suppose that this occurrence of g in t_0 is as indicated by

$$\Gamma, Z \vdash t_0 = L[d], \quad \Gamma, Z, \vec{v} \vdash d = \text{case } gpq \text{ of } (\dots),$$

where \vec{v} is the local environment for $L[-]$. Writing \star for $[\vec{z} \mapsto \vec{r}]$, we have $\ll \Pi(t_0, \xi) \gg = \ll t_0^{\star} \gg$; and if $C[-]$ is the context encapsulating the remainder of the plugging $\Pi(e, \xi)$, then we may write

$$\Gamma \vdash K[c] = t = \ll C[\Pi(t_0, \xi)] \gg = \ll C[t_0^{\star}] \gg = \ll C[L^{\star}[\ll d^{\star} \gg]] \gg,$$

where

$$\ll d^{\star} \gg = \text{case } g \ll p^{\star} \gg \ll q^{\star} \gg \text{ of } (\dots).$$

We claim that we are in the situation of Lemma 19, taking Γ, K, d, K', c of the lemma to be respectively $\Gamma, C[L^*[-]], \ll d^* \gg, K, c$. Condition 1 of the lemma holds because $C[t_0^*]$ is constructed by substitution from normal-form terms of level $\leq k$, and conditions 2 and 3 are immediate in the present setup.

By Lemma 19, we may therefore conclude that for a suitable substitution $[\vec{v} \mapsto \vec{s}]$ with $\Gamma, \vec{v}' \vdash \vec{s}$ regular, $\Gamma \vdash \ll \ll d^* \gg [\vec{v} \mapsto \vec{s}] \gg$ is head-spinal. (Note that the local variables of $C[-]$ do not appear in d^* , because $\Gamma, Z \vdash t_0$ and $\Gamma \vdash \vec{r}$.) Equivalently, we may say that $\ll d[\vec{v}^+ \mapsto \vec{s}^+] \gg$ is head-spinal, where

$$[\vec{v}^+ \mapsto \vec{s}^+] = [\vec{z} \mapsto \vec{r}, \vec{v} \mapsto \vec{s}],$$

so that $\Gamma, \vec{v}' \vdash \vec{s}^+$ and $\text{lv}(\vec{v}^+), \text{lv}(\vec{v}') \leq k$. (Note that the \vec{z} do not appear free in \vec{s} , nor the \vec{v} in \vec{r} .)

Since $\Gamma, Z, V \vdash d$ and $\Gamma, \vec{v}' \vdash \vec{s}^+$ are regular, we are in the situation of Lemma 20, so may conclude that d itself is spinal, and hence that t_0 is spinal. We have thus shown that k -plugging cannot assemble spinal terms from non-spinal ones, and this completes the proof. \square

In particular, since the procedure $g \vdash F_{k+1}[g]$ mentioned at the start of the section is spinal, we may conclude that this procedure is not PCF_k^Ω -denotable, and hence neither is the procedure Y_{k+1} . This establishes Theorem 5.

In the following section, we shall exploit the fact that Theorem 21 also holds relative to the modified notion of spinal term appropriate to a variable $g : 0 \rightarrow (k+1) \rightarrow (k+1)$. Finally, we shall use the fact that this theorem also holds for an innocuous extension of PCF_k^Ω with a constant $\text{byval} : (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N} \rightarrow \mathbb{N}$, whose denotation in SP^0 we take to be

$$\lambda f x. \text{case } x \text{ of } (i \Rightarrow \text{case } fi \text{ of } (j \Rightarrow j)) .$$

To see this, it suffices to add an extra clause to the inductive proof of Theorem 21 noting that the procedure for byval is non-spinal. (The significance of the operator byval is discussed extensively in [14].)

5 Non-definability in the extensional model

To obtain corresponding non-definability results for SF rather than SP^0 , one must show not only that the canonical procedures Y_τ considered above are not PCF_k -definable, but also that no extensionally equivalent procedures are. It is easy to see that there are indeed many other procedures Z with the same extension as Y_τ . To give a trivial example, we may present the canonical procedure Y_{k+1} as $\lambda g x. C[g, x]$, where

$$C[g, x] = \text{case } A[g, x] \text{ of } (i \Rightarrow i) , \quad A[g, x] = g(\lambda x'. C[g, x'])x^\eta .$$

However, another candidate for the fixed point operator is

$$Z_0 = \lambda g x. \text{case } A[g, x] \text{ of } (i \Rightarrow C[g, x]) .$$

Intuitively, this computes the desired value twice, discarding the first result.

As a slightly more subtle example, consider the procedure

$$Z_1 = \lambda g x. \text{case } g(\lambda x'. \text{case } A[g, x] \text{ of } (i \Rightarrow C[g, x'])) x^n \text{ of } (k \Rightarrow k) .$$

Here, within the $\lambda x'$ subterm, we have smuggled in a repetition of the top-level computation $A[g, x]$ before proceeding to evaluate what is really required. The effect is that $\lambda x'. \text{case } A[g, x] \text{ of } (i \Rightarrow C[g, x'])$ may be extensionally below $\lambda x'. C[g, x']$, and this may indeed affect the result when g is applied. However, this can only happen when $Y_{k+1}gx$ is undefined anyway, so it is easy to see that Z_1 as a whole will have the same extension as Y_{k+1} .

Yet another way to construct procedures extensionally equivalent to Y_{k+1} is to vary the subterms of the form x^n (where x has type k). For instance, in the case $k = 1$, we could replace x^n by an extensionally equivalent term such as

$$X_0 = \lambda y^0. x(\lambda. \text{case } y \text{ of } (0 \Rightarrow \text{case } x(\lambda. 0) \text{ of } (j \Rightarrow 0) \mid i + 1 \Rightarrow i + 1)) .$$

This is different in character from the previous examples: rather than simply repeating the computation of xy^n , we are performing the specific computation $x(\lambda. 0)$ which we can see to be harmless given that this point in the tree has been reached. Clearly, such ‘time-wasting’ tricks as the above may be combined and elaborated to yield more complex examples of procedures equivalent to Y .

However, all of the above are rather innocuous variations and do not really yield a fundamentally different method of computing fixed points. For example, the bodies of both Z_0, Z_1 are still head-spinal terms, and it is essentially the spines that are really computing the desired fixed point by the canonical method. This suggests that we should try to show that every procedure extensionally equivalent to Y_{k+1} or $Y_{0 \rightarrow (k+1)}$ is spinal; from this it would follow easily by Theorem 21 that the fixed point functional Y_{k+1} or $Y_{0 \rightarrow (k+1)}$ in SF is not definable in PCF_k^Ω .

Unfortunately, we are currently unable to show this in the case of Y_{k+1} : indeed, the syntactic analysis of procedures $Z \approx Y_{k+1}$ appears to present considerable technical difficulties. We shall establish the result for the case of $Y_{0 \rightarrow (k+1)}$, although even here, it is easiest to concentrate not on $Y_{0 \rightarrow (k+1)}$ itself, but on a certain functional Φ that is readily definable from it.

Specifically, within PCF_{k+1} , let us define

$$\begin{aligned} \Phi & : (0 \rightarrow (k+1) \rightarrow (k+1)) \rightarrow (0 \rightarrow (k+1)) \\ \Phi g^{0 \rightarrow (k+1) \rightarrow (k+1)} & = Y_{0 \rightarrow (k+1)} (\lambda f^{0 \rightarrow (k+1)}. \lambda n. g n (f(\text{suc } n))) , \end{aligned}$$

so that informally

$$\Phi g n = g(n, g(n+1, g(n+2, \dots))) .$$

(This generalizes the definition of the functional F mentioned at the end of Section 2.) For each $n \in \mathbb{N}$, let $g \vdash \Phi_n[g] = \Phi g \hat{n} : k+1$, and let p_n denote the

canonical NSP for $\Phi_n[g]$ (that is, the one arising from the above PCF definition via the standard interpretation in \mathbf{SP}^0). These procedures may be defined simultaneously by:

$$g \vdash p_n = \lambda x^k. \text{case } g(\lambda.n) p_{n+1} x^\eta \text{ of } (i \Rightarrow i) : k+1.$$

By a syntactic analysis of the possible forms of (simple) procedures $g \vdash q \approx p_n$, we will show that any such q is necessarily spinal. Here we have in mind the modified notion of spinal term that is applicable to terms involving a global variable $g : \rho$, where $\rho = 0 \rightarrow (k+1) \rightarrow (k+1)$ (see the explanation following Definition 18). Using Theorem 21 (understood relative to this modified setting), it will then be easy to conclude that within \mathbf{SF} , the element $\llbracket \lambda g. \Phi_0[g] \rrbracket$, and hence $Y_{0 \rightarrow (k+1)}$ itself, is not PCF_k^Ω -definable in \mathbf{SF} .

To show that any $q \approx p_n$ is head-spinal, our approach will be as follows. First, we show that any such q must broadly resemble p_n in at least its top-level structure, in that q must have the form $\lambda x. \text{case } garo \text{ of } (\dots)$, where the arguments a, r, o are closely related to the corresponding arguments $(\lambda.n), p_{n+1}, x^\eta$ occurring within p_n . We do this by showing that if q were to deviate in any way from this prescribed form, we would be able to cook up procedures $G \in \mathbf{SP}^0(\rho)$ and $X \in \mathbf{SP}^0(k)$ manifesting an extensional difference between q and p_n , i.e. such that $q[g \mapsto G]X \not\approx p_n[g \mapsto G]X$. (Contrary to our usual convention, we will here use the uppercase letters G, X to range over normal-form closed procedures that may be substituted for g, x respectively.) In particular, we shall establish a sufficiently close relationship between r and p_{n+1} that the same analysis can then be iterated to arbitrary depth, showing that q has a spinal structure as required.

The main complication is that r need not superficially resemble p_{n+1} , since within r , the crucial application of g that effectively computes the value of p_{n+1} may be preceded by other ‘time-wasting’ applications of g (the idea is illustrated by the example Z_1 above). However, it turns out that such time-wasting subterms $ga^1r^1o^1$ must be of a certain kind if the extensional behaviour $q \approx p_n$ is not to be jeopardized: in particular, the first argument a^1 must evaluate to some $i < n$. (As in the example of Z_1 , the idea is that if the subterm $ga^1r^1o^1$ merely repeats some ‘outer’ evaluation, it will make no overall difference to the extension if the evaluation of this subterm does not terminate.) In order to formulate the relationship between r and p_{n+1} , we therefore need a means of skipping past such time-wasting applications in order to reach the application of g that does the real work. This is achieved with the help of a *masking* operator $\mu_{n,n'}$, which (for any $n \leq n'$) overrides the behaviour of g on numerical arguments $n \leq i < n'$ with a trivial behaviour returning the dummy value 0.

We now proceed to our formal development. As a brief comment on notation, we recall from Section 2 that the relations \approx and \preceq of observational equivalence and inequality make sense not just for elements of \mathbf{SP}^0 but for arbitrary meta-terms (including applications), closed or otherwise. Throughout this section, for typographical convenience, we will tend to express the required relationships

mostly at the level of meta-terms, writing for instance $pq \approx n$ rather than the equivalent $\ll pq \gg = \lambda.n$ or $p \cdot q = \lambda.n$. We shall also perpetrate other mild abuses of notation, such as writing a procedure $\lambda.n$ simply as n (except for special emphasis), $\lambda\vec{x}.\perp$ as \perp , x^η as x , a meta-expression **case** A **of** $(i \Rightarrow i)$ just as A , and abbreviating a substitution $[g \mapsto G, x \mapsto X]$ to $[G, X]$.

We shall say that $G \in \mathbf{SP}^0(0 \rightarrow (k+1) \rightarrow (k+1))$ is *strict* if $G \perp ro \approx \perp$ for any r, o . Clearly, G is strict iff $G \approx \lambda izx. \mathbf{case} \ i \ \mathbf{of} \ (j \Rightarrow G(\lambda.j)z^\eta x^\eta)$. In connection with meta-terms with free variable g , we shall write $T \approx' T'$ to mean that $T[g \mapsto G] \approx T'[g \mapsto G]$ for all *strict* G ; the notation \preceq' is used similarly. We shall actually analyse the syntactic forms of procedures $g \vdash q$ based on the assumption that $q \succeq' p_n$.

We shall say a procedure $g \vdash q$ is *simple* if for every application *garo* appearing within q , the first argument a is a numeral $\lambda.n$. The following observation simplifies our analysis of terms somewhat; it uses the operator *byval* introduced at the very end of Section 4.

Proposition 22 *If there is a procedure $g \vdash q \succeq' p_n$ -denotable in \mathbf{PCF}_k^Ω , then there is a simple procedure $g \vdash q' \succeq' p_n$ denotable in $\mathbf{PCF}_k^\Omega + \mathbf{byval}$.*

PROOF Suppose $g \vdash q$ is \mathbf{PCF}_k^Ω -denotable where $q \succeq' p_n$, and write

$$S[g] = \lambda izx. \mathbf{case} \ i \ \mathbf{of} \ (j \Rightarrow g(\lambda.j)z^\eta x^\eta) .$$

Take $g \vdash q' = \ll q[g \mapsto S[g]] \gg$; it is easy to check that q' is denotable in $\mathbf{PCF}_k^\Omega + \mathbf{byval}$. Then $q \approx' q'$ since $S[G] \approx G$ for all strict G , so $q' \succeq' p_n$. Finally, q' is clearly simple: every occurrence of g within $q[g \mapsto S[g]]$ has a numeral as its first argument, so the same will be true of $\ll q[g \mapsto S[g]] \gg$. \square

For any $n \leq n'$, let us define the *masking* $\mu_{n,n'}(g)$ of g to be the following procedure term:

$$g^\rho \vdash \mu_{n,n'}(g) = \lambda izx. \mathbf{case} \ i \ \mathbf{of} \ (n \Rightarrow 0 \mid \dots \mid n' - 1 \Rightarrow 0 \mid - \Rightarrow gixx) : \rho .$$

We may also write $\mu_{n,n'}(P)$ for $\mu_{n,n'}(g)[g \mapsto P]$ if P is any meta-procedure of type ρ . We write $\mu_{n,n+1}$ simply as μ_n ; note also that $\mu_{n,n}(g) \approx' g$. Clearly $\mu_n(\mu_{n+1}(\dots(\mu_{n'-1}(g))\dots)) \approx \mu_{n,n'}(g)$.

We shall say that a meta-term $P : \rho$ is *trivial at* n if $P(\lambda.n)\perp\perp \approx 0$. Note that $\mu_{n,n'}(G)$ is trivial at each of $n, \dots, n' - 1$ for any G ; indeed, G is trivial at $n, \dots, n' - 1$ iff $G \succeq \mu_{n,n'}(G)$.

The following lemma now implements our syntactic analysis of the top-level structure of simple procedures $q \succeq' p_n$.

Lemma 23 *Suppose $g \vdash q$ is simple and $q \succeq' p_n$. Then q has the form $\lambda x^k. \mathbf{case} \ \mathbf{garo} \ \mathbf{of} \ (\dots)$, where:*

1. $a = \lambda.n$,
2. $o[g \mapsto G] \succeq x^\eta$ whenever G is trivial at n ,

3. $r[g \mapsto \mu_n(g), x \mapsto X] \succeq' p_{n+1}$ for any X .

PROOF Suppose $q = \lambda x^k.e$. Clearly e is not constant since $q \succeq p_n$; and if e had head variable x , we would have $q[g \mapsto \lambda izx. \text{case } i \text{ of } (- \Rightarrow 0)](\lambda w.\perp) = \perp$, whereas $p_n[g \mapsto \lambda izx. \text{case } i \text{ of } (- \Rightarrow 0)](\lambda w.\perp) = 0$, contradicting $q \succeq' p_n$. So e has the form $\text{case } garo \text{ of } (\dots)$.

For claim 1, we have $a = \lambda.m$ for some $m \in \mathbb{N}$ because q is simple. Suppose that $m \neq n$, and consider

$$G' = \lambda izx. \text{case } i \text{ of } (n \Rightarrow 0 \mid - \Rightarrow \perp) .$$

Then for any X , clearly $q[G']X \approx \perp$, whereas $p_n[G']X \approx G'(\lambda.n)(\dots)X \approx 0$, contradicting $q \succeq' p_n$. Thus $a = \lambda.n$.

For claim 2, suppose that $G(\lambda.n)\perp\perp \approx 0$ but not $o[G] \succeq x^\eta$; then we may take $X \in \mathbf{SP}^0(k)$ and $u \in \mathbf{SP}^0(k-1)$ such that $Xu \approx l \in \mathbb{N}$ but $o[G, X]u \not\approx l$. Now define

$$G' = \lambda izx. \text{case } i \text{ of } (j \Rightarrow \text{case } xu \text{ of } (l \Rightarrow Gjzx \mid - \Rightarrow \perp)) .$$

Then $G' \preceq G$, so $o^*u \not\approx l$ where $*$ = $[G', X]$; hence $G'a^*r^*o^* \approx \perp$ and so $q[G']X \approx \perp$. On the other hand, we have

$$\begin{aligned} p_n[G']X &\approx \text{case } G'(\lambda.n)p_{n+1}^*X \text{ of } (i \Rightarrow i) \\ &\approx \text{case } Xu \text{ of } (l \Rightarrow G(\lambda.n)p_{n+1}^*X) \approx 0 , \end{aligned}$$

contradicting $q \succeq' p_n$. Thus $o[G] \succeq x^\eta$.

For claim 3, suppose that $p_{n+1}[G]X' \approx l$ for some strict $G \in \mathbf{SP}^0(\rho)$ and $X' \in \mathbf{SP}^0(k)$. We wish to show that $r[\mu_n(G), X]X' \approx l$ for any X . Suppose not, and consider

$$G' = \lambda izx. \text{case } i \text{ of } (n \Rightarrow \text{case } zX' \text{ of } (l \Rightarrow 0 \mid - \Rightarrow \perp) \mid - \Rightarrow Gizx) .$$

Then $G' \preceq \mu_n(G)$, so $r[G', X]X' \not\approx l$. Moreover, since $a = \lambda.n$ by claim 1, we see that $G'a^*r^*o^* \approx \perp$, where $*$ = $[G', X]$, so that $q[G']X \approx \perp$. On the other hand, we have

$$\begin{aligned} p_n[G']X &\approx \text{case } G'(\lambda.n)p_{n+1}^*X \text{ of } (i \Rightarrow i) \\ &\approx \text{case } p_{n+1}^*X' \text{ of } (l \Rightarrow 0) . \end{aligned}$$

Here, since p_{n+1} does not contain x free, we have $p_{n+1}^* = p_{n+1}[G']$. But it is easy to see that $p_{n+1}[G'] \approx p_{n+1}[G]$, since every occurrence of g within p_{n+1} is applied to $\lambda.n'$ for some $n' > n$, and for all such n' we have $G'(\lambda.n') \approx G(\lambda.n')$. (Since p_{n+1} contains infinitely many applications of g , an appeal to continuity is formally required here.) But $p_{n+1}[G]X' \approx l$ by assumption; thus $p_{n+1}^*X' \approx l$, allowing us to complete the proof that $p_n[G']X \approx 0$. Once again, this contradicts $q \succeq' p_n$, so claim 3 is established. \square

In the light of Appendix A, one may strengthen claim 2 of the above lemma by writing $o[g \mapsto G] \approx x^\eta$. This gives a fuller picture of the possible forms of terms $q \approx p_n$, but is not needed for showing that such q are spinal.

We have now completed a circle, in the sense that claim 3 tells us that $\ll r[g \mapsto \mu_n(g), x \mapsto X] \gg$ itself satisfies the hypothesis for q (with $n+1$ in place of n), and so can be iteratively subjected to the same analysis. However, it still remains to see what this analysis tells us about the syntactic form of r itself, as distinct from $\ll r[g \mapsto \mu_n(g)] \gg$. (In the light of claim 3, the variable x may be safely ignored here.)

Lemma 24 *Suppose $g \vdash q$ is simple and $q[g \mapsto \mu_{n,n'}(g)] \succeq' p_{n'}$. Then q has the form $\lambda x^k. E[\text{case garo of } (\dots)]$, where:*

1. $E[-]$ has empty local variable environment,
2. $a = \lambda.n'$,
3. $o[g \mapsto G] \succeq x^\eta$ whenever G is trivial at n, \dots, n' ,
4. $r[g \mapsto \mu_{n,n'+1}(g), x \mapsto X] \succeq' p_{n'+1}$ for any X .

PROOF Let $q' = \ll q[g \mapsto \mu_{n,n'}(g)] \gg$. Under the above hypotheses, we have by Lemma 23 that q' is of the form $\lambda x. \text{case } ga'r'o' \text{ of } (\dots)$, where $a' = \lambda.n'$, $o'[g \mapsto G] \succeq x^\eta$ whenever G is trivial at n' , and $r'[g \mapsto \mu_{n'}(g)] \succeq' p_{n'+1}$. Write q as $\lambda x. E[\text{case garo of } (\dots)]$ where the displayed occurrence of g originates the head g of q' via the substitution $^\dagger = [g \mapsto \mu_{n,n'}(g)]$.

Suppose that the hole in $E[-]$ appeared within an abstraction $\lambda y. -$; then the hole in $E[g \mapsto \mu_{n,n'}(g)][-]$ would likewise appear within such an abstraction. Moreover, the evaluation of $q[g \mapsto \mu_{n,n'}(g)]$ consists simply of the contraction of certain expressions $\mu_{n,n'}(g)(\lambda.m)r''o''$ to either 0 or $g(\lambda.m)r''o''$, followed by some reductions $\text{case } 0 \text{ of } (i \Rightarrow e_i) \rightsquigarrow e_0$; thus, any residue in q' of the critical g identified above will likewise appear underneath λy . But this is impossible, because the head g of q' is a residue of this g by assumption. This establishes condition 1.

In the light of this, by Lemma 19(i) we have $a' \approx a^\dagger$, $o' \approx o^\dagger$ and $r' \approx r^\dagger$. But since q is simple, a is a numeral, so $a = \lambda.n'$, giving condition 2. For condition 3, suppose G is trivial at n, \dots, n' . Then $G \succeq \mu_{n,n'+1}(G)$, so that

$$o[g \mapsto G] \succeq o[g \mapsto \mu_{n,n'}(\mu_{n'}(G))] \approx o^\dagger[g \mapsto \mu_{n'}(G)] \approx o'[g \mapsto \mu_{n'}(G)] \succeq x^\eta,$$

since $\mu_{n'}(G)$ is trivial at n' . Condition 4 also holds since $r'[g \mapsto \mu_{n'}(g)] \approx r^\dagger[g \mapsto \mu_{n'}(g)] \approx r[g \mapsto \mu_{n,n'+1}(g)]$, where $r'[g \mapsto \mu_{n'}(g)] \succeq' p_{n'+1}$. \square

Corollary 25 *If $g \vdash q$ is simple and $q \succeq' p_n$, then q is spinal (in the modified sense).*

PROOF Since condition 3 of the above lemma matches its hypotheses, starting from the assumption that $q \approx' q[g \mapsto \mu_{n,n}(g)] \succeq' p_n$, we may apply the lemma iteratively to obtain a spinal structure as prescribed by Definition 18 (subject to the relevant adjustments for $g : 0 \rightarrow (k+1) \rightarrow (k+1)$). \square

Thus, if there exists a PCF_k^Ω -denotable procedure $t \approx \lambda g. p_0$, for instance, then by Proposition 22 there is also a simple such procedure t' denotable in

$\text{PCF}_k^\Omega + \text{byval}$, and by Corollary 25, this t' will be spinal in the modified sense. But this contradicts Theorem 21 (understood relative to the modified setting, and applied to $\text{PCF}_k^\Omega + \text{byval}$ as indicated at the end of Section 4). We conclude that no $t \approx \lambda g.p_0$ can be PCF_k^Ω -denotable. Since the interpretation of PCF_k^Ω in \mathbf{SF} factors through \mathbf{SP}^0 , this in turn means that within the model \mathbf{SF} , the element $\llbracket \lambda g.\Phi_0[g] \rrbracket$ is not definable in PCF_k^Ω . On the other hand, this element is obviously definable relative to $Y_{0 \rightarrow (k+1)} \in \mathbf{SF}$ even in PCF_1 , so the proof of Theorem 2 is complete.

6 Extensional non-definability of Y_{k+1}

We have so far shown that the element $Y_{0 \rightarrow (k+1)} \in \mathbf{SF}$ is not PCF_k^Ω -definable. To complete the picture, we shall now refine our methods slightly to show that even $Y_{k+1} \in \mathbf{SF}$ is not definable. Since $\bar{k} + \bar{1}$ is clearly a retract of every level $k+1$ type, this will establish that no $Y_\sigma \in \mathbf{SF}$ where $\text{lv}(\sigma) = k+1$ is definable in PCF_k^Ω .

The idea is as follows. Within each type level $k \geq 1$, we can stratify the types into *sublevels* (k, l) where $l = 1, 2$, essentially by taking account of the ‘width’ of the type as well as its depth. We thus obtain a sublanguage $\text{PCF}_{k,l}^\Omega$ of PCF_k^Ω by admitting Y_σ only for types σ of sublevel (k, l) or lower. Any putative PCF_k^Ω term that defined $Y_{k+1} \in \mathbf{SF}$ would then be a term of some $\text{PCF}_{k,l}^\Omega$.

Our previous argument showed that $Y_{k+1} \in \mathbf{SP}^0$ is not denotable in $\text{PCF}_{k,l}^\Omega$ (using the fact that \bar{k} is not a pseudo-retract of a finite product of level $k-1$ types), and hence that $Y_{0 \rightarrow (k+1)}$ is not $\text{PCF}_{k,l}^\Omega$ -denotable. However, there is actually plenty of spare headroom between $\text{PCF}_{k,l}^\Omega$ and the pure type $k+1$ which we can exploit: specifically, for a certain type $\rho_{k,l+1}$ of sublevel $(k, l+1)$ we may show that $Y_{\rho_{k,l+1}} \in \mathbf{SP}^0$ is not $\text{PCF}_{k,l}^\Omega$ -denotable, essentially because the relevant type $\rho_{k-1,l+1}$ is not a pseudo-retract of a finite product of sublevel $(k-1, l)$ types. (A slightly different argument is needed for the case $k=1$.) For non-definability in \mathbf{SF} , we again need to move up to $0 \rightarrow \rho_{k,l+1}$, but this is still within level k and so is easily seen to be a retract of $\bar{k} + \bar{1}$. It follows that $Y_{k+1} \in \mathbf{SF}$ is not $\text{PCF}_{k,l}^\Omega$ -denotable (where l is arbitrary).

The following definition sets out the more fine-grained stratification of types that we shall use.

Definition 26 (i) The width $w(\sigma)$ of a type σ is defined inductively as follows:

$$w(\mathbf{N}) = 0, \quad w(\sigma_0, \dots, \sigma_{r-1} \rightarrow \mathbf{N}) = \max(r, w(\sigma_0), \dots, w(\sigma_{r-1})).$$

For $k \geq 1$, we say σ has sublevel (k, l) if $\text{lv}(\sigma) = k$ and $w(\sigma) = l$. If $\sigma = \mathbf{N}$, we simply say that σ has sublevel 0. We order sublevels in the obvious way: 0 is the lowest sublevel, and $(k, l) < (k', l')$ if either $k < k'$ or $k = k'$ and $l < l'$.

(ii) For each k, l we define a type $\rho_{k,l}$ by:

$$\rho_{0,l} = \mathbf{N}, \quad \rho_{k+1,l} = \rho_{k,l}, \dots, \rho_{k,l} \rightarrow \mathbf{N} \quad (l \text{ arguments}).$$

When $k \geq 1$, we may call $\rho_{k,l}$ the homogeneous type of sublevel (k, l) .

The following facts are easily established. Here we shall say that σ is a *simple retract* of τ if there is a PCF₀-definable retraction $\sigma \triangleleft \tau$ within \mathbf{SP}^0 .

Proposition 27 (i) For $k \leq 1$, every type of sublevel (k, l) or lower is a simple retract of $\rho_{k,l}$. Hence (for all $k \geq 0$) for every finite list of types σ_i of level $\leq k$, there is some l such that each σ_i is a simple retract of $\rho_{k,l}$.

(ii) Every finite product of level $\leq k$ types is a simple retract of $\overline{k+1}$.

(iii) If σ is a simple retract of τ , then Y_σ is PCF₀-definable from Y_τ in \mathbf{SP}_0 .

PROOF (i) The first claim (for $k \leq 1$) is easy by induction on k , and the second claim (which is trivial when $k = 0$) follows easily.

(ii) By induction on k . The case $k = 0$ is easy. For $k \geq 1$, suppose $\sigma_0, \dots, \sigma_{m-1}$ are level $\leq k$ types, where $\sigma_i = \tau_{i0}, \dots, \tau_{i(n_i-1)} \rightarrow \mathbb{N}$ for each i . Here the τ_{ij} are of level at most $k-1$, so by (i), we may choose l such that each τ_{ij} is a simple retract of $\rho_{k-1,l}$. Taking $n = \max_i n_i$, we then have that each σ_i is a simple retract of $\rho_{k-1,l}, \dots, \rho_{k-1,l} \rightarrow \mathbb{N}$ (with n arguments). The product $\Pi_i \sigma_i$ is therefore a simple retract of the type $\sigma = \mathbb{N}, \rho_{k-1,l}, \dots, \rho_{k-1,l} \rightarrow \mathbb{N}$. But by the induction hypothesis, the product of $\mathbb{N}, \rho_{k-1,l}, \dots, \rho_{k-1,l}$ is a simple retract of \bar{k} whence σ itself is a simple retract of $\bar{k+1}$.

We leave (iii) as an exercise. \square

Next, we adapt the proof of Theorem 6 to establish the crucial gap between $\rho_{k,l}$ and $\rho_{k,l+1}$. This gives an indication of how our methods may be used to map out the embeddability relation between types in finer detail, although we leave an exhaustive investigation of this to future work.

Theorem 28 Suppose $k \geq 1$. Within \mathbf{SF} , the type $\rho_{k,l+1}$ is not a pseudo-retract of any finite product of types of sublevel $\leq (k, l)$ or lower.

PROOF In view of Proposition 27(i), it will suffice to show that $\rho_{k,l+1}$ is not a pseudo-retract of a finite power of $\rho_{k,l}$. We argue by induction on k . The arguments for both the base case $k = 1$ and the step case $k > 1$ closely parallel the argument for the step case in Theorem 6, so we treat these two cases together as far as possible, omitting details that are easy adaptations of those in the earlier proof.

Suppose for contradiction that there were procedures

$$z^\rho \vdash t_i : \sigma \quad (i < m), \quad x_0^{\sigma_0}, \dots, x_{m-1}^{\sigma_{m-1}} \vdash r : \rho,$$

where $\rho = \rho_{k,l+1}$, $\sigma = \rho_{k,l}$, such that $z \vdash r[\vec{x} \mapsto \vec{t}] \succeq z^\eta$. Let $v = \ll r[\vec{x} \mapsto \vec{t}] \gg$, so that $z \vdash v$. As in the proof of Theorem 6, one may show that v has the form $\lambda f_0 \dots f_l. \text{case } z p_0 \dots p_l \text{ of } (\dots)$ where $p_i[z \mapsto \lambda \vec{w}. 0] \succeq f_i$ for each i . Next, we note that $r[\vec{x} \mapsto \vec{t}]$ reduces in finitely many steps to a head normal form $\lambda f_0 \dots f_l. \text{case } z P_0 \dots P_l \text{ of } (\dots)$ where $\ll P_i \gg = p_i$ for each i ; moreover, the ancestor of the leading z here must lie within some t_i , say at the head of some subterm $z \vec{P}'$, where \vec{P}' is an instance of \vec{P} via some substitution \dagger .

At this point, the arguments for the base case and step case part company. In the base case $k = 1$, we have that $z^\rho \vdash t_i : \sigma$ where $\sigma = \mathbb{N}^{(l)} \rightarrow \mathbb{N}$; thus there are

no bound variables within t_i except the top-level ones — say w_0, \dots, w_{l-1} , all of type \mathbf{N} . So in fact \dagger has the form $[\vec{w} \mapsto \vec{W}]$ for certain meta-terms $f_0^{\mathbf{N}}, \dots, f_l^{\mathbf{N}}, z \vdash W_j : \mathbf{N}$. Now consider the terms $f_0, \dots, f_l \vdash \vec{W}^*$ and $\vec{w} \vdash \vec{P}'^*$, writing $*$ for the substitution $[z \mapsto \lambda w.0]$. These compose to yield $\vec{f} \vdash \ll \vec{P}^* \gg = \ll \vec{p}^* \gg \succeq f$, so we have expressed \mathbf{N}^{l+1} as a pseudo-retract of \mathbf{N}^l within \mathbf{SF} . As already noted in the course of the proof of Theorem 6, this is impossible.

For the step case $k > 1$, we proceed much as in the proof of Theorem 6, using the substitution \dagger to express $\rho_{k-1,l+1}$ as a retract of a finite product of types of sublevel $(k-1, l)$ or lower, contrary to the induction hypothesis. We leave the remaining details as an exercise. \square

We now outline how the ideas of Sections 3, 4 and 5 may be adapted to show that $Y_{\rho_{k,l+1}} \in \mathbf{SP}^0$ is not $\mathbf{PCF}_{k,l}^\Omega$ -denotable. We assume that $k \geq 2$ for the time being (the case $k = 1$ will require special treatment). The adaptations are mostly quite systematic: the type $\rho_{k,l+1}$ now plays the role of $\overline{k+1}$; types of sublevel $\leq (k, l)$ play the role of types of level $\leq k$; $\rho_{k-1,l+1}$ plays the role of \overline{k} ; and types of sublevel $\leq (k-1, l)$ play the role of types of level $< k$. Since the proof we are adapting is quite lengthy, we leave many routine details to be checked by the reader.

First, the evident adaptation of Definition 12 yields a notion of k, l -plugging where the plugging variables are required to be of sublevel $\leq (k, l)$, and we thus obtain an inductive characterization of the $\mathbf{PCF}_{k,l}^\Omega$ -denotable procedures analogous to Theorem 14.

We may now proceed to the ideas of Section 4, taking g to be a fixed global variable of type $\rho_{k,l+1} \rightarrow \rho_{k,l+1}$.

Definition 29 *An environment Γ is (k, l) -regular if Γ contains g and all other variables in Γ are of sublevel $\leq (k, l)$. A meta-term-in-environment $\Gamma \vdash T$ is regular if Γ is regular and all variables bound within T are of sublevel $\leq (k, l)$.*

Our convention here will be that Γ ranges over regular environments, and Roman capitals V, Z range over sets of variables of sublevel $\leq (k, l)$. The notion of spinal term carries over as follows:

Definition 30 (i) *If \vec{x} is a list of variables of type $\rho_{k-1,l+1}$ and V a set of variables of sublevel $\leq (k, l)$, a substitution $^\circ = [\vec{w} \mapsto \vec{r}]$ is called \vec{x}, V -closed if the r_i contain no free variables, except that if $w_i \in V$ and w_i is of sublevel $\leq (k-1, l)$ then r_i may contain the \vec{x} free.*

(ii) *We coinductively declare a regular $\Gamma \vdash e$ to be (k, l) -head-spinal w.r.t. \vec{x}, V iff e has the form $\mathbf{case} \ g(\lambda \vec{x}'. E[e']) \vec{o} \ \mathbf{of} \ (\dots)$, where $E[-]$ is an expression context, and*

- *for some \vec{x}, V -closed specialization $^\circ$ covering the free variables of \vec{o} other than those of \vec{x} , we have $\vec{o}^\circ \succeq \vec{x}^\eta$,*
- *e' is head-spinal w.r.t. \vec{x}', V' , where V' is the local variable environment for $E[-]$.*

(iii) We say a regular term $\Gamma \vdash t$ is (k, l) -spinal if it contains a head-spinal subexpression w.r.t. some \vec{x}, V .

Lemma 19 and its proof now go through with the above adaptations; here the local environments \vec{v}, \vec{v}' are now of sublevel $\leq (k, l)$, and that part (iii) of the lemma now states that if $K[-]$ contains no redexes with operator of sublevel $> (k, l)$, then the substitution † is trivial for variables of sublevel $\geq (k-1, l+1)$. The crucial Lemma 20, which forms the heart of our proof, now translates as follows:

Lemma 31 *Suppose we have k, l -regular terms*

$$\Gamma, \vec{v} \vdash d = \text{case } gpq \text{ of } (\cdots), \quad \Gamma, \vec{v}' \vdash \vec{s},$$

where \vec{v}, \vec{v}' are of sublevel $\leq (k, l)$, and $\Gamma, \vec{v}' \vdash \ll d[\vec{v} \mapsto \vec{s}] \gg$ is k, l -head-spinal with respect to some \vec{x}, V . Then d itself is k, l -spinal.

The entire proof of this lemma translates systematically according to the correspondences we have indicated, invoking Theorem 28 for the fact that $\rho_{k-1, l+1}$ is not a pseudo-retract of a product of sublevel $\leq (k-1, l)$ types. The analogue of Theorem 21 now goes through readily, so we obtain:

Theorem 32 *If $k \geq 2$, then every $\text{PCF}_{k, l}^\Omega$ -denotable procedure is non- k, l -spinal. \square*

As in our original proof, we will actually use the version of this theorem for the modified notion of spinal term incorporating the extra argument b , and for the extension of $\text{PCF}_{k, l}^\Omega$ with the operator *byval*.

To adapt the material of Section 5, we take g to be a variable of type $0 \rightarrow \rho_{k, l+1} \rightarrow \rho_{k, l+1}$, and argue that the $\text{PCF}_{k, l+2}$ -definable element $\Phi = \lambda g. Y_{0 \rightarrow \rho_{k, l+1}}(\lambda f n. g n (f(suc n)))$ within \mathbf{SF} is not $\text{PCF}_{k, l}^\Omega$ -definable. The proof is a completely routine adaptation of that in Section 5. Since $Y_{0 \rightarrow \rho_{k, l+1}}$ is readily definable from Y_{k+1} by Proposition 27, this implies that $Y_{k+1} \in \mathbf{SF}$ is not $\text{PCF}_{k, l}^\Omega$ -definable. Since l was arbitrary, we have shown:

Theorem 33 *For $k \geq 2$, the element $Y_{k+1} \in \mathbf{SF}$ is not definable in PCF_k .*

A somewhat different approach is needed to cover the case $k = 1$. This is because at level 0 our only type is \mathbf{N} , so we are unable to make a distinction between sublevels l and $l+1$. To establish Lemma 31 in this case, we again wish to show that we cannot pass in the content of the relevant \vec{x}' to the relevant s_i , but now the idea is to appeal to the fact that \vec{x}' consists of $l+1$ variables of type \mathbf{N} , whereas s_i accepts at most l arguments of type \mathbf{N} . (We have already seen that \mathbf{N}^{l+1} cannot be a retract of \mathbf{N}^l .) However, we also need to exclude the possibility that the substitution $^\circ$ is being used to import some components of \vec{x}' . We can achieve this if we require $^\circ$ to be actually closed rather than just \vec{x}', V' -closed, and it turns out that this is permissible if we also tighten our notion of spinal term slightly, essentially to ensure that no intermediate applications of g appear in between those declared to constitute the spine of the term:

Definition 34 (i) A substitution $\circ = [\vec{w} \mapsto \vec{r}]$ is called *closed* if the r_i contain no free variables.

(ii) We coinductively declare a regular $\Gamma \vdash e$ to be *strongly head-spinal* w.r.t. \vec{x}, V iff e has the form **case** $g(\lambda \vec{x}'. E[e']) \vec{o}$ of (\dots) , where $E[-]$ is an expression context, and

- the hole within $E[-]$ does not itself occur within an application gpq ,
- for some closed specialization \circ covering the free variables of \vec{o} other than those of \vec{x} , we have $\vec{o}^\circ \succeq \vec{x}^\eta$,
- e' is head-spinal w.r.t. \vec{x}', V' , where V' is the local variable environment for $E[-]$.

The notion of *strongly spinal term* follows suit.

The counterpart of Lemma 19 goes through as expected, although without part (iii): the relevant sublevel distinction does not exist at type level 0, and we cannot conclude that the substitution in question is trivial for all variables of type N. We may now indicate the required changes to Lemma 31 and its proof:

Lemma 35 Suppose we have $1, l$ -regular terms

$$\Gamma, \vec{v} \vdash d = \text{case } gpq \text{ of } (\dots), \quad \Gamma, \vec{v}' \vdash \vec{s},$$

where \vec{v}, \vec{v}' are of sublevel $\leq (1, l)$, and $\Gamma, \vec{v}' \vdash \ll d[\vec{v} \mapsto \vec{s}] \gg$ is strongly $1, l$ -head-spinal with respect to some \vec{x}, V . Then d itself is strongly $1, l$ -spinal.

PROOF The proof of Lemma 20 up to the end of the proof of Claim 1 adapts straightforwardly, and is somewhat simplified by the fact that the substitution \circ is closed. As sketched above, the crucial contradiction is provided by the fact that \mathbb{N}^{l+1} is not a pseudo-retract of \mathbb{N}^l in SF.

For the remainder of the proof, the key point to note is that \vec{v}'' (the local environment for $C[-]$) is actually empty in this case. This is because $C[-]$ is in normal form and contains no free variables of level ≥ 2 except g , so any λ -term containing the hole would need to appear as an argument to g . It would then follow that the hole within $E[-]$ lay within an argument to some occurrence of g , as precluded by the definition of strongly spinal term.

It follows trivially that the \vec{x}', \vec{v}'' -closed substitution \circ' constructed at the very end of the proof is actually closed. Moreover, the spinal structure of d' identified by the proof cannot contain any intermediate applications of g , since these would give rise under evaluation to intermediate applications of g in the spine of $\ll d[\vec{v} \mapsto \vec{s}] \gg$ as precluded by Definition 34. Thus, the identified spinal structure in d' is actually a strongly spinal structure, and the argument is complete. \square

A trivial adaptation of the proof of Theorem 21 now yields:

Theorem 36 No $\text{PCF}_{1,l}^\Omega$ -denotable procedure can be strongly $1, l$ -spinal. \square

From here on, we again follow the original proof closely. The only additional point to note is that in place of Corollary 25 we now require that any simple $g \vdash q$ with $q \succeq' p_n$ must be strongly spinal, and this is in fact easily seen from the proof of Lemma 24. We therefore have everything we need for:

Theorem 37 $Y_2 \in \text{SF}$ is not definable in PCF_1 . \square

7 Related and future work

7.1 Other hierarchies of Y combinators

Previous results to the effect that Y combinators for level $k + 1$ are in some way not definable from those for level k have been obtained by Damm [6] and Statman [22]. It is convenient to discuss the latter of these first.

Statman works in the setting of the simply typed λY -calculus, essentially the pure λ -calculus extended with constants $Y_\sigma : (\sigma \rightarrow \sigma) \rightarrow \sigma$ and reduction rules $Y_\sigma M \rightsquigarrow M(Y_\sigma M)$. He gives a succinct proof that Y_{k+1} is not definable from Y_k up to computational equality, based on the following idea. If Y_{k+1} were definable from Y_k , it would follow that the recursion equation $Y_{k+1}g = g(Y_{k+1}g)$ could be derived using only finitely many uses of the equation $Y_k M = M(Y_k)M$ (say m of them). It would then follow, roughly speaking, that mn recursion unfoldings for Y_k would suffice to fuel n recursion unfoldings for Y_{k+1} . On the other hand, it can be shown that the size of normal-form terms definable using n unfoldings of Y_{k+1} (as a function of the size of the starting term) grows faster than can be accounted for with mn unfoldings of Y_k .

The language λY is seemingly less powerful than PCF ,⁵ although this is perhaps not the most essential difference between Statman’s work and ours. More fundamentally, Statman’s method establishes the non-definability only up to computational equality (that is, the equivalence relation generated by the reduction rules), whereas we have been concerned with non-definability modulo observational (or extensional) equivalence. Even for non-definability in SP^0 , it would seem an approach along Statman’s lines would be unlikely to yield much information, since there is no reason why the number of unfoldings of Y_k required to generate the NSP for Y_{k+1} to depth n should not grow dramatically as a function of n . Nonetheless, it is striking that methods so markedly different from ours can be used to obtain closely related results.

A result very similar to Statman’s was obtained earlier in Damm [6], but by a much more indirect route as part of a far-reaching investigation of the theory of tree languages. In Damm’s setting, programs are *recursion schemes*—essentially, families of simultaneous (and possibly mutually recursive) defining equations in typed λ -calculus—but in essence these can be considered as terms of λY relative to some signature consisting of typed constants. Any such program can be expanded to an infinite tree (essentially a kind of Böhm tree),

⁵One might consider translating PCF into λY by representing natural numbers as Church numerals; however, it appears that predecessor is not λY -definable for this representation.

and Damm’s result (Theorem 9.8 of [6]) is that up to tree equality, there are programs definable by level $k + 1$ recursions but not level k ones. The notion of tree equality here is more generous than computational equality (making Damm’s result somewhat stronger than Statman’s), although still much more fine-grained (in the case of a signature for PCF) than equality in \mathbf{SP}^0 , let alone in \mathbf{SF} . It therefore seems unlikely, once again, that an approach to our theorems from this direction will be forthcoming.

What is certainly of interest, however, is the connection that Damm establishes between programming-style recursion schemes on the one hand, and hierarchies of language families on the other. Specifically, Damm establishes a close connection between tree grammars and λ -calculus, showing that the languages inhabiting level k of the *OI-hierarchy*⁶ are exactly those that can be generated by level k recursion schemes. He also shows that the OI-hierarchy is strict, and it is from this that the superiority of level $k + 1$ schemes over level k ones is deduced. Thus, whilst the statement of Damm’s theorem on the recursion scheme hierarchy has a syntactic flavour (it is formulated in terms of tree equality), the theorem is reached via an essentially ‘semantic’ investigation of the expressive power of such schemes within a particular domain, namely that of language definitions. Of course, this is a long way from the setting of computable functionals over \mathbb{N}_\perp that we consider in the present paper,⁷ and it is not clear whether there is any substantial connection to be made here, but the applications of higher-type recursions in language theory are certainly intriguing.

7.2 Relationship to game semantics

Next, we comment briefly how our work relates to the known *game models* of PCF [1, 7]. It is known that these models are in fact isomorphic to our \mathbf{SP}^0 , although the equivalence between the game-theoretic definition of application and our own is mathematically non-trivial (the situation is discussed in [14, Section 7.4]). This raises the obvious question of whether our proofs could be conducted equally well, or better, in a game-semantical setting. Whilst a direct translation is presumably possible, our present impression is that sequential procedures, and our calculus of meta-terms in particular, allows one to see the wood for the trees more clearly, and also to draw more easily on familiar intuitions from λ -calculus. However, a closer look at game-semantical approaches would be needed in order to judge whether either approach really offers some genuine mathematical advantage over the other.

⁶This is a language hierarchy whose first two levels are the sets of regular and context-free languages respectively. OI stands for *outermost-innermost*.

⁷One noteworthy difference is that even at base type, more objects are definable by $(k + 1)$ -recursions than k -recursions in Damm’s setting, whereas for us, all base type objects are of course trivially definable in \mathbf{PCF}_0 .

7.3 Sublanguages of PCF: further questions

We now turn our attention to some potential extensions and generalizations of our work.

So far, we have worked mainly with a coarse stratification of types in terms of their levels, although we have illustrated in Section 6 how finer stratifications are also possible. Naturally, there is scope for a still more fine-grained analysis of types and the relative strength of their Y combinators; this is of course closely related to the task of mapping out the embeddability relation between types (as in Subsection 2.4) in finer detail.

Even at level 1, there is some interest here. Our analysis in Section 6 has shown that, for $l \geq 1$,

- the element $Y_{N^{l+1} \rightarrow N} \in \mathbf{SP}^0$ is not \mathbf{PCF}_0^Ω -definable from $Y_{N^l \rightarrow N}$,
- the element $Y_{N^{l+2} \rightarrow N} \in \mathbf{SF}$ is not \mathbf{PCF}_0^Ω -definable from $Y_{N^l \rightarrow N}$.

However, this leaves us with a small gap for \mathbf{SF} : e.g. we have not shown either that $Y_{N \rightarrow N}$ is strictly weaker than $Y_{N^2 \rightarrow N}$ or that $Y_{N^2 \rightarrow N}$ is strictly weaker than $Y_{N^3 \rightarrow N}$, although according to classical logic, at least one of these must be the case. (This is reminiscent of some well-known situations from complexity theory.) We expect that a more delicate analysis will allow us to fill this gap. One can also envisage even more fine-grained hierarchy obtained by admitting other base types such as the booleans or unit type.

A closely related task is to obtain analogous results for the *call-by-value* interpretation of the simple types (as embodied in Standard ML, for example). As is shown in [14, Section 4.3], a call-by-value (partial) type structure \mathbf{SF}_v can be constructed by fairly general means from \mathbf{SF} : here, for example, $\mathbf{SF}_v(\bar{1})$ consists of all partial functions $N \multimap N$ rather than (monotone) total functions $N_\perp \rightarrow N_\perp$, and $\mathbf{SF}_v(\bar{2})$ consists of partial functions $N_\perp^N \multimap N$. From known results on the interencodability of call-by-name and call-by-value models (see [14, Section 7.2]), it is easy to read off the analogue of Corollary 3 for $\mathbf{SF}_v^{\text{eff}}$; however, more specific results on the relative strengths of various Y combinators within \mathbf{SF}_v would require more detailed reworking of our arguments.

Of course, one can also pose relative definability questions for other elements besides Y combinators. For instance, it is natural to consider the *higher-order primitive recursors* rec_σ of System T, as well as the closely related *iterators* $\text{iter}_{\sigma\tau}$ which we touched on in Section 2.5:

$$\begin{aligned} \text{rec}_\sigma & : \quad \sigma \rightarrow (\sigma \rightarrow N \rightarrow \sigma) \rightarrow N \rightarrow \sigma , \\ \text{iter}_{\sigma\tau} & : \quad (\sigma \rightarrow (\sigma + \tau)) \rightarrow \sigma \rightarrow \tau . \end{aligned}$$

We have already noted that if all iterators are available then it is easy to define all primitive recursors, but that \mathbf{PCF}_0^Ω extended with iterators for all types does not suffice to define even Y_1 . We believe that by working with a suitably chosen substructure of \mathbf{SP}^0 , it should be possible to strengthen our results to show that \mathbf{PCF}_k^Ω extended with all iterators does not suffice to define Y_{k+1} . The dual question, roughly speaking, is whether any or all of the recursors

rec_σ or iterators $iter_{\sigma\mathbb{N}}$ for types σ of level $k + 1$ are definable in PCF_k . We conjecture that they are not, and that this can be shown by suitably choosing a substructure of \mathbf{SP}^0 so as to exclude such $iter_{\sigma\mathbb{N}}$. (This would incidentally answer Question 2 of [4, Section 5].) One could also look for substructures that more precisely determine the strength of various *bar recursion* operators or the *fan functional*. All in all, our experience leads us to expect that many further substructures of \mathbf{SP}^0 should be forthcoming, leading to a harvest of non-definability results exhibiting a rich ‘degree structure’ among PCF-computable functionals.

Another very natural kind of question is the following: given a particular sublanguage \mathcal{L} of PCF, what is the *simplest* possible type for an element of \mathbf{SF}^{eff} that is not definable in \mathcal{L} ? Or to look at it another way: given a type σ , what is the smallest natural sublanguage of PCF that suffices for defining all elements of $\mathbf{SF}^{\text{eff}}(\sigma)$? Here the analysis of [14, Section 7.1] yields several positive definability results, whilst the analysis of the present paper provides ammunition on the negative side. The current state of our knowledge is broadly as follows. As in [14], we write \mathbf{Klex}^{\min} of the language of *Kleene μ -recursion*: this is equivalent (in its power to define elements of \mathbf{SF}) to PCF_0 extended with a strict primitive recursor for ground type and a strict iterator for ground type, but with no form of general recursion.

- For first-order types σ , even the language \mathbf{Klex}^{\min} suffices for defining all elements of $\mathbf{SF}^{\text{eff}}(\sigma)$; likewise, the oracle version $\mathbf{Klex}^{\min^\Omega}$ suffices for $\mathbf{SF}(\sigma)$.
- For second-order types σ of the special form $(\mathbb{N} \rightarrow \mathbb{N})^r \rightarrow \mathbb{N}$, $\mathbf{Klex}^{\min^\Omega}$ still suffices for $\mathbf{SF}(\sigma)$; however, this result is non-constructive, and \mathbf{Klex}^{\min} does not suffice for $\mathbf{SF}^{\text{eff}}(\sigma)$. (This follows from recent unpublished work of Dag Normann.)
- For general second-order types, $\mathbf{Klex}^{\min^\Omega}$ no longer suffices, but the languages PCF_1 , PCF_1^Ω suffice for \mathbf{SF}^{eff} , \mathbf{SF} respectively—indeed, even the single recursion operator $Y_{\mathbb{N} \rightarrow \mathbb{N}}$ is enough here.
- For third-order types, we do not know whether PCF_1 suffices (for \mathbf{SF}^{eff}). We do know that PCF_2 suffices, and that $Y_{\mathbb{N} \rightarrow \mathbb{N}}$ alone is not enough.
- For types of level $k \geq 4$, PCF_{k-3} does not suffice, but PCF_{k-2} does.

Again, there is scope for a more fine-grained view of the hierarchy of types.

7.4 Other models and languages

We have so far concentrated almost entirely on PCF-style sequential computation. To conclude, we briefly consider which other notions of higher-order computation are likely to present us with an analogous situation.

As already noted at the end of Subsection 2.4, several extensions of PCF studied in the literature present a strikingly different picture: in these settings, universal types exist quite low down, and as a consequence, only Y combinators of low type (along with the other constants of the language) are required for full definability. There is, however, one important language which appears to be more analogous to pure PCF in these respects, namely an extension with *local state* (essentially Reynolds’s *Idealized Algol*). This language was studied in [2], where a fully abstract game model was provided (consisting of well-bracketed but possibly non-innocent strategies). Unpublished work by Jim Laird has shown that there is no universal type in this setting. We would conjecture also that the recursion hierarchy for this language is strict; this would constitute a significant strengthening of our present results.

Related questions also arise in connection with *total* functionals. Consider, for example, the type structure \mathbf{Ct} of total continuous functionals, standardly constructed as the extensional collapse (relative to \mathbb{N}) of the Scott model \mathbf{PC} of partial continuous functionals. It is shown by Normann [16] that every effective element of \mathbf{Ct} is represented by a PCF-*definable* element of \mathbf{PC} , and the proof actually shows that \mathbf{PCF}_1 suffices here. (The further generalization of these ideas by Longley [12] actually makes mild use of second-order recursions as in \mathbf{PCF}_2 ; we do not know whether these can be eliminated.) Thus, in this setting, only recursions of low type are needed to obtain all computable functionals. Similar remarks apply to the total type structure \mathbf{HEO} , obtained as the extensional collapse of \mathbf{PC}^{eff} .

On the other hand, one may consider the *Kleene computable* functionals over \mathbf{Ct} , or over the full set-theoretic type structure \mathbf{SS} , as defined by the schemes S1–S9. As explained in [14, Chapter 6], sequential procedures can be seen as abstracting out the algorithmic content common to both PCF-style and Kleene-style computation (note that Kleene’s S9 in some sense does duty for the Y combinators of PCF). This naturally suggests that our strict hierarchy for PCF may have an analogue for the Kleene computable functionals (say over \mathbf{SS} or \mathbf{Ct}), where at level k we consider the evident restriction of S9 to types of level $\leq k$. We conjecture that this is indeed the case, although the required counterexamples may be more difficult to find given that we are limited to working with total functionals.

Appendix A: Super-identity procedures

In the course of our proof, we have frequently encountered assertions of the form $p \succeq x^\eta$ for various procedures $x^k \vdash p$. Although not necessary for our main argument, it is natural to ask whether there are any such procedures other than those for which $p \approx x^\eta$. The following theorem shows that the answer is no: in other words, no procedure $\lambda x.p : k \rightarrow k$ can extensionally ‘improve on’ the identity function. We present this as a result of some interest in its own right, whose proof is perhaps less trivial than one might expect.

Recall that \preceq denotes the extensional order on \mathbf{SF} , as well as the associated

preorder on \mathbf{SP}^0 . Within \mathbf{SF} , we will write $f \prec f'$ to mean $f \preceq f'$ but $f \neq f'$; we also write $f \# f'$ to mean that f, f' have no upper bound with respect to \preceq .

We call an element of \mathbf{SP}^0 *finite* if it is a finite tree once branches of the form $i \Rightarrow \perp$ have been deleted. We say an element of \mathbf{SF} is *finite* if it is represented by some finite element of \mathbf{SP}^0 . We write $\mathbf{SP}^{0,\text{fin}}(\sigma), \mathbf{SF}^{\text{fin}}(\sigma)$ for the set of finite elements in $\mathbf{SP}^0(\sigma), \mathbf{SF}(\sigma)$ respectively.

Theorem 38 (i) If $f \in \mathbf{SF}^{\text{fin}}(k)$ and $f \prec f'$, then there exists $f'' \# f'$ with $f \prec f''$.

(ii) If $\Phi \in \mathbf{SF}(k \rightarrow k)$ and $\Phi \succeq \text{id}$, there can be no $f \in \mathbf{SF}(k)$ with $\Phi(f) \succ f$; hence $\Phi = \text{id}$.

PROOF Both statements are easy for $k = 0, 1$, so we will assume $k \geq 2$.

(i) Suppose $f \prec f'$ where f is finite. Then for some $g \in \mathbf{SF}(k-1)$ we have $f(g) = \perp$ but $f'(g) = n \in \mathbb{N}$, say, and by continuity in \mathbf{SP}^0 we may take g here to be finite. Take $p, q \in \mathbf{SP}^{0,\text{fin}}$ representing f, g respectively; we may assume that p, q are ‘pruned’ so that every subtree containing no leaves $m \in \mathbb{N}$ must itself be a leaf \perp .

Case 1: $g(\perp^{k-2}) = a \in \mathbb{N}$. In this case, we may suppose that $q = \lambda x.a$. Consider the computation of $p \cdot q$. Since all calls to q trivially evaluate to a , this computation follows the rightward path through p consisting of branches $a \Rightarrow \dots$. But since p is finite and $p \cdot q = \perp$ (because $f(g) = \perp$), this path must end in a leaf occurrence of \perp within p . Now extend p to a procedure p' by replacing this leaf occurrence by some $n' \neq n$; then clearly $p' \cdot q = n'$. Taking f'' to be the function represented by p' , we then have $f \preceq f''$ and $f''(g) = n' \# n = f'(g)$, so $f'' \# f'$ (whence also $f'' \neq f$).

Case 2: $g(\perp^{k-2}) = \perp$. Take N larger than n and all numbers appearing in p, q . Define $p' \supseteq p$ as follows: if $p = \lambda x.\perp$, take $p' = \lambda x.N$, otherwise obtain p' from p by replacing each case branch $j \Rightarrow \perp$ anywhere within p by $j \Rightarrow N$ whenever $j \leq N$. Extend q to q' in the same way. Note in particular that every **case** subterm within p', q' will now be equipped with a branch $N \Rightarrow N$.

Now consider the computation of $p \cdot q$. Since p, q are finite and $f(g) = \perp$, this evaluates to an occurrence of \perp which originates from p or q . Since no numbers $> N$ ever arise in the computation, this occurrence of \perp cannot be part of a branch $j \Rightarrow \perp$ for $j > N$, so will have been replaced by N in p' or q' . Now suppose that we head-reduce pq until \perp first appears in head position, and let U be the resulting meta-term. Then it is easy to see that $p'q'$ correspondingly reduces to a meta-term U' with N in head position. (Formally, we reason here by induction on the length of head-reduction sequences not involving the rule for **case** \perp of (\dots) .)

We now claim that $p' \cdot q' = N$. Informally, this is because the head occurrence of N in U' will be propagated to the top level by the case branches $N \Rightarrow N$ within both p' and q' . More formally, let us define the set of meta-expressions *led by* N inductively as follows:

- N is led by N .

- If E is led by N , then so is **case** E of $(i \Rightarrow D_i)$.

We say that an NSP meta-term T is *saturated at N* if every **case** subterm within T has a branch $N \Rightarrow E$ where E is led by N . Clearly $p'q'$ is saturated at N , and it is easy to check that the terms saturated at N are closed under head reductions; thus U' is saturated at N . But we have also seen that U' has N in head position, so is led by N . Finally, an easy induction on term size shows that every finite meta-term that is led by N and saturated at N evaluates to N itself. This shows that $p' \cdot q' = N$.

To conclude, let f'', g' be the functions represented by p', q' respectively, so that $f \preceq f''$ and $g \preceq g'$. Then $f'(g') = n$, but $p'' \cdot q = N$ so $f''(g') = N \neq n$, whence $f'' \not\# f'$ (and also $f'' \neq f$).

(ii) Suppose $\Phi \succeq id$ and $\Phi(f) \succ f$ for some f . Again by continuity, we may take f to be finite. Then by (i), we may take $f'' \succ f$ such that $f'' \# \Phi(f)$. But this is impossible because $\Phi(f'') \succeq f''$ and $\Phi(f'') \succeq \Phi(f)$. Thus $\Phi = id$. \square

It is easy to see that the above theorem holds with any finite type over \mathbb{N} in place of \bar{k} . However, it may trivially fail if the unit type \mathbf{U} is admitted as an additional base type: e.g. the function $(\lambda x. \top) \in \mathbf{SF}(\mathbf{U} \rightarrow \mathbf{U})$ strictly dominates the identity. An interesting question is whether the theorem holds for all finite types over the type \mathbf{B} of booleans: note that the above proof fails here since it requires the base type to be infinite. For comparison, we mention that in other models of computation, improvements on the identity are sometimes possible for such types. For example, if $\sigma = \mathbf{B} \rightarrow \mathbf{B}$, then a functional of type $\sigma \rightarrow \sigma$ strictly dominating the identity exists in the Scott model. Indeed, such a function J can be defined in PCF augmented with the parallel conditional *pif*, e.g. as

$$J = \lambda f^\sigma. \lambda x^{\mathbf{B}}. \text{vote}(f(x), f(tt), f(ff)) .$$

Here *vote* is Sazonov's voting function, definable by

$$\text{vote}(x, y, z) = \text{pif}(x, \text{pif}(y, tt, z), \text{pif}(y, z, ff)) .$$

The point is that J will 'improve' the function $\lambda x. if(x, tt, tt)$ to $\lambda x. tt$. We do not know whether phenomena of this kind can arise within the model \mathbf{SF} .

References

- [1] Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF. *Information and Computation* **163**(2), 409–470 (2000)
- [2] Abramsky, S., McCusker, G.: Game semantics. In: Schwichtenberg, H., Berger, U. (eds.) *Computational Logic: Proceedings of the 1997 Marktoberdorf Summer School*, pp. 1–56. Springer, Heidelberg (1999)
- [3] Amadio, R., Curien, P.-L.: *Domains and Lambda Calculi*. Cambridge Tracts in Theoretical Computer Science 46, Cambridge University Press (1998)

- [4] Berger, U.: Minimisation vs. recursion on the partial continuous functionals. In: Gärdenfors, P., Woleński, J., Kijania-Placek, K. (eds.) *In the Scope of Logic, Methodology and Philosophy of Science: Volume One of the 11th International Congress of Logic, Methodology and Philosophy of Science, Cracow, August 1999*, pp. 57-64. Kluwer, Dordrecht (2002)
- [5] Cartwright, R., Felleisen, M.: Observable sequentiality and full abstraction. In: *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pp. 328-342. ACM Press, New York (1992)
- [6] Damm, W.: The IO- and OI-hierarchies. *Theoretical Computer Science* **20**, 95-207 (1982)
- [7] Hyland, J.M.E., Ong, C.-H.L.: On full abstraction for PCF: I, II, and III. *Information and computation* **163**, 285-408 (2000)
- [8] Jones, N.D.: The expressive power of higher-order types or, life without CONS. *Journal of Functional Programming* **11**, 5-94 (2001)
- [9] Kleene, S.C.: Unimonotone functions of finite types (Recursive functionals and quantifiers of finite types revisited IV). In: A. Nerode and R.A. Shore (eds.), *Proceedings of the AMS-ASL Summer Institute on Recursion Theory*, pp. 119-138. American Mathematical Society, Providence (1985)
- [10] Kristiansen, L.: Higher types, finite domains and resource-bounded Turing machines. *Journal of Logic and Computation* **22(2)**, 281-304 (2012)
- [11] Longley, J.R.: The sequentially realizable functionals. *Annals of Pure and Applied Logic* **117(1)**, 1-93 (2002)
- [12] Longley, J.R.: On the ubiquity of certain total type structures. *Mathematical Structures in Computer Science* **17(5)**, 841-953 (2007)
- [13] Longley, J.: Bar recursion is not $T+\min$ definable. *Informatics Research Report EDI-INF-RR-1420*, University of Edinburgh (2015)
- [14] Longley, J. and Normann, D.: Higher-Order Computability. To appear in 'Computability in Europe' series, Springer (2015-16)
- [15] Milner, R.: Fully abstract models of typed λ -calculi. *Theoretical Computer Science* **4(1)**, 1-22 (1977)
- [16] Normann, D.: Computability over the partial continuous functionals. *Journal of Symbolic Logic* **65(3)**, 1133-1142 (2000)
- [17] Normann, D., Sazonov, V.Yu., The extensional ordering of the sequential functionals. *Annals of Pure and Applied Logic* **163(5)** 575-603 (2012)
- [18] Plotkin, G.D.: LCF considered as a programming language. *Theoretical Computer Science* **5(3)**, 223-255 (1977)

- [19] Sazonov, V.Yu.: Expressibility in D. Scott's LCF language. *Algebra and Logic* **15(3)**, 192–206 (1976)
- [20] Sazonov, V.Yu.: An inductive definition and domain theoretic properties of fully abstract models of PCF and PCF+. *Logical Methods in Computer Science* **3(3)**, 50 pages (2007)
- [21] Scott, D.S.: A type-theoretical alternative to ISWIM, CUCH, OWHY. In: A collection of contributions in honor of Corrado Böhm on the occasion of his 70th birthday. (Edited version of an unpublished note first circulated in 1969.) *Theoretical Computer Science* **121(1-2)**, 411–440 (1993).
- [22] Statman, R.: On the λY calculus. *Annals of Pure and Applied Logic* **130**, 325–337 (2004).